

# A CONTROL THEORETIC PERSPECTIVE ON LEARNING IN ROBOTICS

A Dissertation  
Presented to  
The Academic Faculty

by

Rowland O'Flaherty

In Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy  
in  
Robotics



School of Electrical and Computer Engineering  
Georgia Institute of Technology  
May 2016

Copyright © 2016 by Rowland O'Flaherty

# A CONTROL THEORETIC PERSPECTIVE ON LEARNING IN ROBOTICS

Approved by:

Professor Magnus Egerstedt, Advisor  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Professor Ayanna Howard  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Professor Patricio Vela  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Professor Charles Isbell  
School of Interactive Computing  
*Georgia Institute of Technology*

Professor Jonathan Rogers  
School of Mechanical Engineering  
*Georgia Institute of Technology*

Date Approved: 7 December 2015

*To my wife, Neda Bagheri,*

*for the unconditional support that made this work possible.*

## ACKNOWLEDGEMENTS

First and foremost, I would like to thank my doctoral advisor, Dr. Magnus Egerstedt, for his unmatched ability to disseminate his wealth of knowledge through outstanding course lectures, precise attention to detail, and focused one-on-one meetings. All the skills I have developed as a robotist directly result from Magnus' sheer comprehension of the field of control theory and his talent and enthusiasm for teaching and learning.

Next, I would like to thank my committee for taking the time to review and guide my doctoral research. Much of the work in my research was made possible by the knowledge I acquired from their courses and one-on-one discussions.

Finally, the countless interactions with my fellow GRITS lab peers comprised some of my most fruitful discussions. It was in this unique environment where I truly took hold of the material needed to conduct my doctoral research. For this reason, I thank each and everyone one of the members of the GRITS lab.

# TABLE OF CONTENTS

<b>DEDICATION</b>	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iv</b>
<b>LIST OF TABLES</b>	<b>viii</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>SUMMARY</b>	<b>xii</b>
<b>I INTRODUCTION</b>	<b>1</b>
<b>II LEARNING OPTIMAL COVERAGE (THEORY)</b>	<b>4</b>
2.1 Introduction	4
2.2 Problem Statement	6
2.3 $E^3$ Algorithm Details	9
2.3.1 Ergodic Trajectory Optimization	10
2.3.2 Initializing Input Trajectory for Ergodic Optimization	13
2.3.3 Updating Estimate of Information Distribution	14
2.3.4 Additional Notes on $E^3$	16
2.4 Exploration vs. Exploitation	17
2.5 Minimizing Ergodicity and Effort	18
2.5.1 Descent Direction Derivation	20
2.6 Feasible Trajectory Projection	21
2.7 Armijo Step Size Calculation	24
2.8 Conclusion	24
<b>III LEARNING OPTIMAL COVERAGE (EXPERIMENTS)</b>	<b>26</b>
3.1 Experimental Setup Introduction	26
3.2 Differential Drive Robot	26
3.2.1 Differential Drive Dynamics	26
3.2.2 Differential Drive Control	28

3.2.3	Differential Drive Robot Experimental Setup . . . . .	31
3.3	Quadrotor Robot Experimental Setup . . . . .	33
3.3.1	Quadrotor Dynamics . . . . .	33
3.3.2	Euler-Lagrange Derivation . . . . .	37
3.3.3	Linearization . . . . .	40
3.3.4	Diffeomorphic State Dynamics . . . . .	40
3.3.5	Quadrotor Control . . . . .	43
3.4	Experimental Results . . . . .	50
3.4.1	Differential Drive Robot Experimental Results . . . . .	51
3.4.2	Quadrotor Experimental Results . . . . .	54
3.5	Conclusion . . . . .	57
<b>IV</b>	<b>LEARNABILITY . . . . .</b>	<b>59</b>
4.1	Introduction to Learnability . . . . .	59
4.2	Learnability Definition . . . . .	61
4.3	A Simple Yet Illustrative Example . . . . .	63
4.4	Learnability for Linear Systems With Quadratic Cost . . . . .	64
4.4.1	Problem Statement . . . . .	65
4.4.2	Linear System With Quadratic Cost Learnability Theorem . . . . .	66
4.4.3	Linear System With Quadratic Cost Learnability Proof . . . . .	67
4.5	Inverted Pendulum Robot . . . . .	73
4.6	Simple Example Revisited On A Real Robot . . . . .	75
4.7	Conclusion . . . . .	79
<b>V</b>	<b>LOW-DIMENSIONAL LEARNING FOR COMPLEX ROBOTS . . . . .</b>	<b>80</b>
5.1	Introduction to Low-Dimensional Learning . . . . .	80
5.2	System Overview . . . . .	83
5.2.1	System Dynamics . . . . .	84
5.2.2	Primitive Controllers and Decision Conditions . . . . .	85
5.2.3	Hybrid System Formulation . . . . .	87

5.2.4	Reward Function . . . . .	89
5.3	Learning Algorithm . . . . .	90
5.3.1	Learning Controller Actions . . . . .	90
5.3.2	Learning Decision Boundaries . . . . .	92
5.3.3	Exploration vs. Exploitation . . . . .	92
5.4	Examples . . . . .	93
5.4.1	Example Simulated System . . . . .	93
5.4.2	Explanation of Results For Simulated System . . . . .	94
5.4.3	Simulated High Dimensional System . . . . .	96
5.4.4	Physical Robotic System . . . . .	98
5.5	Conclusion . . . . .	100
<b>VI</b>	<b>CONCLUSION . . . . .</b>	<b>103</b>
<b>APPENDIX A</b>	<b>— NOTATION . . . . .</b>	<b>105</b>
<b>APPENDIX B</b>	<b>— DERIVATIONS . . . . .</b>	<b>112</b>
<b>REFERENCES</b>	<b>. . . . .</b>	<b>119</b>

## LIST OF TABLES

1	Parameters used to control the Khepera differential drive robot. . . . .	31
2	Variable constant values used in the E <sup>3</sup> algorithm robot exploration experiments.	32
3	Simulation Results: Mean Cost For 100 Trials . . . . .	52
4	External state with different parameters using 100 trials of our learning compared against the optimal state value. . . . .	95



## LIST OF FIGURES

1	Illustration of the ergodic trajectory $x(t)$ with respect to the distribution $\phi(x)$ , where $\int_A \phi(x)dx = \frac{1}{t_f}(t_4 - t_3 + t_2 - t_1)$ . . . . .	5
2	Geodesic distances (normalized by $\frac{ X }{C}$ ) for non-sampled states (shaded areas) from the search area boundaries (black areas) and from other sampled states (white areas) for a given robot trajectory (solid line). . . . .	15
3	Cartoon intuitive representation of how the projection optimization works. Starting at a feasible trajectory (all feasible trajectories line on the thick black line) $z_i$ , the decent direction (green arrow), $\zeta$ , that moves the the trajectory to a minimal cost (contours of cost are shown in thin black lines), which is an infeasible trajectory. The projection operator projects (red dash line) the infeasible trajectory back to the feasible manifold, $z_{i+i}$ . This is repeated until the trajectory reaches the local feasible optimal trajectory at $z^*$ . . . . .	19
4	Results from applying projection operator to simple single integrator system. Plotted is the input (top left), state (top middle), and spatial transform components (top right), histogram of the state trajectory (bottom right), and the desired and actual information distribution (bottom right). . . . .	23
5	Robots used for $E^3$ experiments . . . . .	27
6	Differential drive robot cartoon. The positional portion of the state is at $x$ while the diffeomorphic positional portion of the state is at $\tilde{x}$ . . . . .	28
7	Quadrotor robot cartoon. The positional portion of the state is located at the center of the quadrotor at $\xi$ and the diffeomorphic state is offset from $\xi$ by a distance $d$ . . . . .	36
8	These plots show the response from an LQR and CLF feedback controller to stabilize the attitude of a quad from two different initial conditions. Initial condition 1: roll = 0.6, pitch = 0.6, and yaw = 0.6. Initial condition 2: roll = 1.0, pitch = -0.5, and yaw = 1.0. . . . .	50
9	Experimental results of running the $E^3$ algorithm on a differential drive robot exploring an area with an underlying unknown information distribution. (a) Underlying unknown information distribution and exploration trajectory of the robot (solid line) from the start location (dot) to the final location (X). (b) Distribution of differential drive robot locations over time while exploring the space. . . . .	52

10	Evolution of $E^3$ algorithm implemented on the Khepera differential drive robot exploring a 2-dimensional space with an underlying unknown information distribution. Each sub-figure shows the confidence value of the occupancy grid (black-gray-white regions), information gain map (dotted contour lines), initial trajectory (dotted line), locally optimal trajectory (dashed line), and actual trajectory the robot followed (solid line) from its start location (dot) to stop location (X). . . . .	53
11	Plots of simulation results using the $E^3$ algorithm to have a quadrotor explore a 3D space. All four plots are of the same data from different perspectives. The black line is the quadrotor's trajectory, starting from the green circle and ending at the red cross. . . . .	55
12	Experimental results of running the $E^3$ algorithm on a quadrotor exploring an area with an underlying unknown information distribution. (a) Underlying unknown information distribution and exploration trajectory of the quadrotor (solid line) from the start location (dot) to the final location (X). (b) Distribution of quadrotor locations over time while exploring the space. . . .	56
13	Evolution of $E^3$ algorithm implemented on the quadrotor exploring a space with an underlying unknown information distribution. Each sub-figure shows the confidence value of the occupancy grid (black-gray-white regions), information gain map (dotted contour lines), initial trajectory (dotted line), locally optimal trajectory (dashed line), and actual trajectory the robot followed (solid line) from its start location (dot) to stop location (X). . . . .	57
14	Illustration of two-wheel inverted pendulum robot [12]. . . . .	74
15	The top plot shows the results from experiment 1, which is when the robot is measuring distances in the direction it is moving. The bottom plot shows the results from experiment 2, which is when the robot is not measuring distances in the direction it is moving. The dashed black line is the distance the robot has moved. The dotted light gray line is the accumulated cost. The horizontal line along the bottom of the plots shows when the robot is exploring new learned actions (dark areas of the line) and exploiting previous learned actions (light areas of the line) . . . . .	78
16	Physical robotic system used to test the learning algorithm. . . . .	83
17	Illustration of the internal state $x_I \in \mathbb{R}^2$ being controlled into a limit cycle with controllers $\kappa_1, \dots, \kappa_4$ and with the location of the decision boundaries defined by the parameters $o_i$ and $d_i$ . The flow set $C$ is the interior of the box and the jump set $D$ is the exterior of the triangle formed by $p_1$ , $p_2$ , and $p_3$ . The interior of this triangle must be convex. . . . .	88
18	Learning To Locomote Algorithm . . . . .	91
20	An illustration of the $N$ -link high dimensional simulate serpent system. . . .	96

21	Forward movement verse update number of a 15-link serpent simulated system learning to move forward. After $8 \times 10^4$ updates the system has entered a stable limit cycle and moves forward at the maximum rate. . . . .	98
22	(Top) Plot shows the physical robot learning to move forward. The forward progress is shown (blue (dark) line with plus sign markers) as well as when the robot is exploring (red (dark) marks) and exploiting (green (light) marks). (Bottom) Plot shows the robots movements after learning how to move both forward (line with plus sign markers) and backwards (line with circle markers).	100
19	These plots show the results of running the learning algorithm on Brockett's system. Top: Plot of the internal state components, $x_{I1}$ (blue (dark) line) and $x_{I2}$ (green (light) line), verses time. The internal state constraints are also shown (black dashed lines). Second: Plot of the external state and reward, $x_E$ (orange (dark) line) and $R(x_E, u)$ (blue (light) line), respectively, verses time. Third: Plot of the control state and boundary state, $\xi$ (top pink (dark) line) and $\beta$ (bottom yellow (light) line), respectively, verses time. Bottom: plot of the maximum variance in the Q values for a given state, $\max(\sigma^2(s))$ (purple (dark) line), verses time. The variance threshold for deciding between exploration and exploitation is also shown (straight solid black line). In all the plots it is shown when the learning algorithm is exploring or exploiting with the red (dark) and green (light) marks, respectively, across the bottom of each plot. . . . .	102

## SUMMARY

For robotic systems to continue to move towards ubiquity, robots need to be more autonomous. More autonomy dictates that robots need to be able to make better decisions. Control theory and machine learning are fields of robotics that focus on the decision making process. However, each of these fields implements decision making at different levels of abstraction and at different time scales. Control theory defines low-level decisions at high rates, while machine learning defines high-level decision at low rates. The objective of this research is to integrate tools from both machine leaning and control theory to solve higher dimensional, complex problems, and to optimize the decision making process.

Throughout this research, multiple algorithms were created that use concepts from both control theory and machine learning, which provide new tools for robots to make better decisions. One algorithm enables a robot to learn how to optimally explore an unknown space, and autonomously decide when to explore for new information or exploit its current information. Another algorithm enables a robot to learn how to locomote with complex dynamics. These algorithms are evaluated both in simulation and on real robots. The results and analysis of these experiments are presented, which demonstrate the utility of the algorithms introduced in this work. Additionally, a new notion of “learnability” is introduced to define and determine when a given dynamical system has the ability to gain knowledge to optimize a given objective function.

# CHAPTER I

## INTRODUCTION

For robotic and cyber-physical systems to function properly in the real world, these systems need to make a large number of calculated decisions. These choices range from fast low-level decisions (e.g. stability control) to slow high-level decisions (e.g. path planning). The theory behind the fast low-level decisions generally comes from the field of control theory, while the theory behind the slow high-level decisions generally comes from the field of machine learning. As robotic systems advance and their corresponding tasks become more complex, the line between the fast low-level decisions and the slow high-level decisions becomes blurred. Thus, it is necessary to use tools from both control theory and machine learning for designing the decision making algorithms used by these robotic systems. The objective of this research is to integrate and to use tools from these two areas of robotics to solve higher dimensional, complex problems, and to optimize the process of exploration and exploitation. Specifically, by letting learning algorithms be informed by the underlying dynamics of the system, a number of benefits are obtained, from reduced complexity to systematic trade offs between exploration and exploitation.

The goal of any learning algorithm is to gather information that is useful for the desired objective, then to use that information to complete the objective in the “best” possible way. This goal raises an additional issue all learning algorithms must address: deciding when to gather new information and when to use the information that has already been gathered to complete the objective. This tradeoff is often called “exploitation versus exploration” [23], which is discussed in Chapter 2 and Chapter 5.

The work outlined in Chapter 2 describes how to calculate a control policy for a single agent system to optimally explore an unknown space to gather information that is distributed

over that space. The agent’s goal is to optimally cover the space by gathering the most information with the least amount of effort. The agent must learn the underlying information distribution and then use that knowledge to plan an optimal trajectory to cover the space. Thus, the agent must decide when to explore and when to exploit. The theory outlined in Chapter 2 is tested in simulation and on real robots. The individual robot dynamics, experimental setup, and results are detailed in Chapter 3.

Certain elements of this research can be viewed as a “nail”, while other elements can be viewed as a “hammer”. The application, which describes an optimal coverage problem, is a “nail”; while the theory, which describes how to optimally decide between exploitation and exploration, is a “hammer”. The objective is to integrate the two to develop an optimal “exploitation versus exploration” theory, and to extend beyond solving an optimal coverage problem for a single agent so it can benefit countless related problems in machine learning and control theory that involve exploration and exploitation.

For the agent to optimally explore the unknown space and learn the underlying information distribution, as briefly described above, it is assumed that it is possible for the agent to learn this distribution given the agent’s sensor and actuator capabilities. What if the agent that is tasked to explore a 3 dimensional space is a ground robot with wheels for actuation and a sensor that only works in the plane? Will the agent be able to learn the underlying distribution? What if instead the same robot, has a sensor that samples in the volume instead of just in the plane, now will the robot be able to learn the underlying distribution? It is not clear when learning is possible or when a problem is “learnable” given the objective, the dynamics, the actuation, and the sensors. An attempt to remedy this challenge would require a formal definition of learnability from a system-theoretic vantage-point. In Chapter 4, a “learnability” definition is provided for a dynamical system with a given objective function. Given this definition, a theorem and proof are provided on when a linear system with a quadratic cost is “learnable”.

Another challenge with using learning for control design is that often the complexity of the

system makes it impractical to use standard learning techniques because of limits in memory usage and/or processing time. This issue is often called the “Curse of Dimensionality”, which was created by Richard Bellman in the 1950’s. This curse is the exponential increase of information that must be learned as the number of possible states and actions in the system increases. A learning algorithm can overcome this complexity issue by using a particular choice of discretization presented in Chapter 5, The algorithm uses boundary conditions coupled with sets of primitive control laws to create motions for the locomotion of complex robotic systems. In particular, the presented algorithm learns actions based on boundary states instead of the actual system states, which greatly reduces the amount of learning that must take place.

To illustrate how this learning algorithm may be used, imagine a situation where a roboticist would like to build a robotic caterpillar, without having to (or even knowing how to) explicitly design the control actions to move the robot forward or backward. Instead, the roboticist simply “loads” the presented learning algorithm together with a library of primitive feedback controllers onto the robotic caterpillar. On its own, the robotic caterpillar learns to move forward and backward. This situation is demonstrated on the robotic caterpillar, which is discussed further in Chapter 5.

Taken together, this research seeks to move the field of robotics closer to autonomy, by advancing the underlying tools and employing new applications in the fields of control theory and machine learning.

## CHAPTER II

### LEARNING OPTIMAL COVERAGE (THEORY)

#### 2.1 *Introduction*

This chapter presents an optimal exploration algorithm, *Ergodic Environmental Exploration* ( $E^3$ ), that minimizes the effort to explore an unknown environment with areas of varying degrees of importance. Applications for the use of  $E^3$  include any type of search over an unknown area with an autonomous robot. Example applications include searching for wildfires in a forest, monitoring areas of high crime in a city, monitoring national borders, and military reconnaissance. The  $E^3$  algorithm removes the decision of when to explore for new areas of importance and when to focus on, or exploit, the current estimate of important areas in order to maximize information gain. The algorithm deals with exploration versus exploitation by exploring and exploiting simultaneously.

The  $E^3$  algorithm is a finite receding horizon optimal control algorithm for exploring and obtaining information in regions with an unknown information distribution<sup>1</sup>. The  $E^3$  algorithm builds off and modifies the *Ergodic Exploration for Distributed Information* (EEDI) algorithm developed by the Neuroscience and Robotics (NxR) Lab at Northwestern University [38, 39, 50]. The EEDI algorithm is an iterative, receding horizon, optimal control algorithm for controlling a mobile sensor to optimally acquire data.  $E^3$  uses the concept from the EEDI algorithm that in each iteration an optimal trajectory is calculated that minimizes the control effort and the ergodic cost with respect to some spatial distribution over the system's state space. The optimization of an ergodic trajectory with respect to effort was developed by the NxR Lab [37].

---

<sup>1</sup>In this work a *distribution* refers to a function with finite support, always positive, and integrates to one.



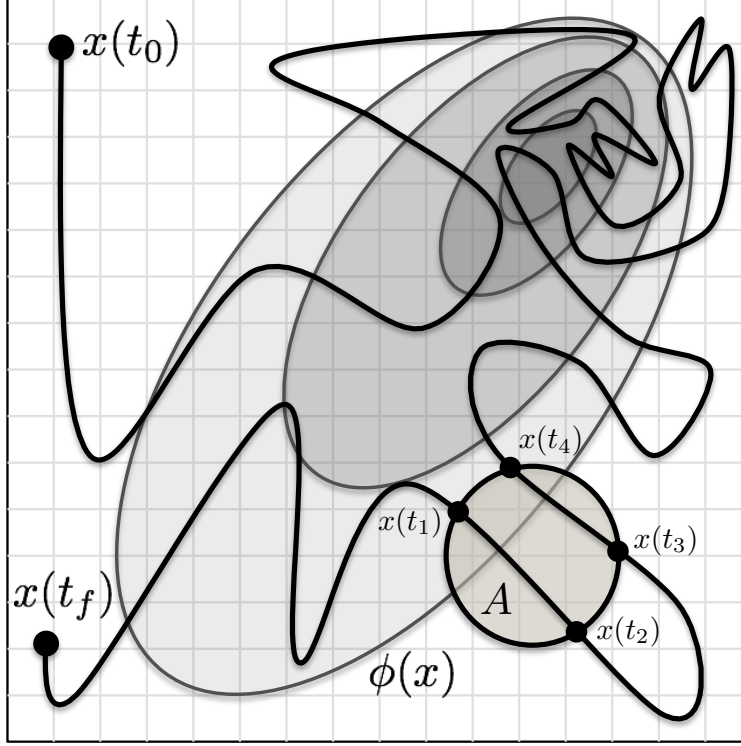


Figure 1: Illustration of the ergodic trajectory  $x(t)$  with respect to the distribution  $\phi(x)$ , where  $\int_A \phi(x)dx = \frac{1}{t_f}(t_4 - t_3 + t_2 - t_1)$ .

The ergodic cost relates the time-averaged behavior of a dynamical system's state trajectory to some spatial distribution. Mathew and Mezic [34] formulated this ergodicity metric for trajectories to define how well the points of a trajectory produced by a dynamical system cover a domain of interest. As shown in [37], the concept of an ergodic dynamical system to a distribution is expressed in Fig. 1. The distribution  $\psi(x)$ , depicted as level sets over the domain  $X$ , is sampled by a sensor following the trajectory  $x(t)$  over the time interval  $T = [t_0, t_f]$ . The trajectory  $x(t)$  is ergodic with respect to the distribution  $\psi(x)$  if the percentage of time spent in any subset  $A$  of  $X$  from  $t = t_0$  to  $t = t_f$  is equal to the measure of  $A$ . For a dynamical system to be ergodic for the distribution  $\phi(x)$ , the condition shown for subset  $A$  must hold for all possible subsets in  $X$ .

The novelty of the  $E^3$  algorithm is the distribution function used in the ergodic optimization. The  $E^3$  algorithm assumes the underlying information distribution is stored in

an occupancy grid across the search space, where each dimension in the grid has  $C$  cells. The use of an occupancy grid is common practice in robot perception and navigation problems [14]. The distribution function in the  $E^3$  algorithm is comprised of the robot’s estimate of the information distribution and its uncertainty in that estimate. Differently, the EEDI algorithm assumes that the underlying information distribution is parameterized by  $\ell$  unknown parameter values. The EEDI solves for the  $\ell$  parameters using the expected value of Fisher information matrix, which is an  $\ell \times \ell$  matrix.

Using the EEDI algorithm to solve for the locally optimal trajectory with an occupancy grid in an  $n$ -dimensional space and  $C$  cells in each dimension, requires  $\ell = C^n$  parameters; and thus, the Fisher information matrix has  $C^{2n}$  elements. The size of the Fisher information matrix becomes infeasible for a distributions modeled with an occupancy grid that has a large number of cells. Additional differences in the  $E^3$  algorithm and the EEDI algorithm include the generation of the initial trajectory used in each iteration of the algorithm and the process in which the information distribution is updated from iteration to iteration.

Other work in exploration of unknown environments involves a variety of methods, including reinforcement learning [29], extended information filters [32] [57], Markov decision processes [1] [6], coverage controllers [20], and model predictive control [15]. The  $E^3$  algorithm stands out from these other methods by focusing on control actions that ensure exploration of the space while also minimizing the control effort to do so.

Experiments were performed to demonstrate the  $E^3$  algorithm’s ability to have a mobile reconnaissance robot optimally explore a 2-dimensional map with non-uniform regions of importance. The experiment details and results are discuss in Chapter 3.

## 2.2 *Problem Statement*

The problem statement that the  $E^3$  algorithm solves is defined in the following section and is different from what has been done in previous work on this subject. In a space of interest,  $X \subset \mathbb{R}^n$ , it is known that there are regions in  $X$  with varying degrees of importance that may

change over time; any further knowledge of these regions is initially unknown. The regions of varying degrees of importance at time  $t$  can be viewed as an information distribution function,  $\psi_t(x)$ , for all  $x \in X$ .

A mobile robot is used to explore and estimate the underlying information distribution. The robot can move through the space and sample the unknown information distribution for a specific region at a single time instance. Moreover, it is desirable to minimize the effort needed to do the exploration.

The robot moves according to its dynamics, which are assumed deterministic and modeled in the general form of

$$\dot{x} = f(x, u), \quad (1)$$

where  $x \in X$  is the state of the robot and  $u \in U \subset \mathbb{R}^m$  is the input to the robot.

As the robot moves through the space  $X$  and samples a set of values from the information distribution,  $\psi_t$ , its state is assumed to be known. In this work it is assumed that the robot's sensor has a delta disc footprint of the form<sup>2</sup>

$$D(x; \delta) = \{x' : \|x' - x\| \leq \delta\}. \quad (2)$$

In other words, the robot is able to measure the exact values from the information distribution,  $\psi_t(x)$ , for all the values in a radius  $\delta$  of its state  $x$  at a given time instant. That being the case, for each position  $x$ , the robot can update a set of values in its current estimate of the information distribution,  $\hat{\psi}_t$ , with a set of measured values from the true distribution,  $\psi_t$ , that correspond to the states in  $D(x; \delta)$ .

The robot is also capable of keeping track of the states it has previously sampled. This capability is encoded in the indicator function  $I_t(x) : X \rightarrow \{0, 1\}$ , which returns a zero if the robot has never measured a value for the state  $x$  before time  $t$ , and a one otherwise. As

---

<sup>2</sup> $x'$  is used to represent an arbitrary state in  $X$ .

the robot moves, it updates the function  $I_t(x)$  with the following update law,

$$I_{t+\Delta t}(x') = \begin{cases} 1 & : x' \in D(x; \delta) \\ I_t(x') & : x' \notin D(x; \delta) \end{cases}, \quad (3)$$

where  $\Delta t$  is the measurement sampling time.

The robot can improve on its estimate of the information distribution for states outside  $D(x; \delta)$  with an update law. In this work the update law for the estimated information distribution function,  $\hat{\psi}_t(x')$ , when the robot is in state  $x$  at time  $t$  is defined by a two step process:

$$\begin{aligned} \text{Step 1: } \hat{\psi}^\dagger(x') &= \begin{cases} \psi_t(x') & : x' \in D(x; \delta) \\ \hat{\psi}_t(x') & : x' \notin D(x; \delta) \end{cases}, \\ \text{Step 2: } \hat{\psi}_{t+\Delta t}(x') &= \begin{cases} \hat{\psi}^\dagger(x') & : x' \in D(x; \delta) \\ \Upsilon(x', \hat{\psi}^\dagger, I_{t+\Delta t}) & : x' \notin D(x; \delta) \end{cases}, \end{aligned} \quad (4)$$

where  $\Upsilon$  is the update function for values of the information distribution that are outside  $D(x; \delta)$ . The  $\Upsilon$  used in the E<sup>3</sup> algorithm is detailed in Section 2.3.3. The initial value of the estimated information distribution is defined with the function  $\hat{\psi}_0(x)$ .

In addition to maintaining an estimate of the information distribution, in this work it is assumed that the robot maintains confidence values for its estimate of the information distribution. Right after the robot has measured a value from the true information distribution it has a high confidence in its estimate of those measurement locations and assigns those locations a confidence value of one. This confidence value exponentially decreases in time. This exponentially decrease in the confidence value allows for the algorithm to adopt to a changing information distribution.

The confidence for a particular state  $x$  at a time  $t$  is encoded in the function  $G_t(x)$ . The update law for the confidence function with the robot at state  $x$  and time  $t$  is,

$$G_{t+\Delta t}(x') = \begin{cases} 1 & : x' \in D(x; \delta) \\ (1 - \frac{\Delta t}{\tau}) G_t(x') & : x' \notin D(x; \delta) \end{cases}. \quad (5)$$

Thus, the update law in (5) means that for each state that is not measured, the confidence value for that state exponentially decays with a time constant of  $\tau$ . This exponential decay is commonly seen in machine learning algorithms and is sometimes known as the “learning rate”. The time constant should be chosen with respect to the time constant of the dynamics of the underlying information distribution. In this work, the information distribution is constant but a non-infinite time constant is used to demonstrate the utility of the confidence function if the information distribution were not constant. The initial confidence value is  $G_0(x')$ .

The goal of the robot is to find an input and state trajectory that minimize the effort and maximizes the cumulative sum of information and confidence over time and space. This goal promotes the robot to visit areas of greatest possible information gain more often than places with lower possible information gain. Areas with high information gain are areas with high uncertainty (i.e. low confidence) and high information content. A function to represent areas of high uncertainty and high information content is

$$H(x) = (1 - G(x))\psi(x), \quad (6)$$

where  $1 - G(x)$  represents uncertainty and  $\psi$  represents information content.

### **2.3 $E^3$ Algorithm Details**

The  $E^3$  algorithm (shown in Algorithm 1) drives the robot to explore the space  $X$ , which will maximize information gain and minimize effort. The algorithm initializes the estimate of the information distribution as a uniform distribution,

$$\hat{\psi}_0(x') = \frac{1}{|X|} \quad \forall x' \in X. \quad (7)$$

In this way, all states are assumed equally important. In addition, the initial confidence value is set to zero

$$G_0(x') = 0 \quad \forall x' \in X, \quad (8)$$

meaning there is zero confidence in the initial estimate of the information distribution.

In each iteration of the algorithm a locally optimal trajectory for the information gain map,  $H_i(x)$ , is updated based on the current estimate of the information distribution and the current confidence value of that estimate. With the updated information gain map, an initial trajectory,  $u_i^0(t)$ , is calculated that visits  $l$  number of “peaks” of the information gain map (see Section 2.3.2). This initial trajectory is not optimized for effort or ergodicity, it is a heuristic to “seed” the ergodic optimization algorithm with a good initial guess at the trajectory. Empirically, this method for generating an initial trajectory has shown to provide stable continuous exploration over the entire space.

Next, the locally optimal ergodic state trajectory,  $x_i^*(t)$ , and locally optimal input trajectory,  $u_i^*(t)$ , are updated based on  $u_i^0(t)$  and  $H_i(x)$  for a time horizon duration of  $t_D$  (see Section 2.3.1) after which the locally optimal trajectories are executed. While executing the locally optimal trajectory, the robot measures the true information distribution at each time step and updates the confidence values and estimated information distribution values (see Section 2.3.3). The iteration repeats, incrementing the iteration counter,  $i$ . This process takes place until the final time of the exploration,  $t_F$ .

Each component of the algorithm is described in more detail in the following subsections. As described in Section 2.1, the  $E^3$  algorithm builds off of the EEDI algorithm; however, there are several differences between the  $E^3$  algorithm and the EEDI algorithm. These differences are also discussed in more detail in the following subsections.

### 2.3.1 Ergodic Trajectory Optimization

It is ideal to have a trajectory for the robot where the amount of time that the robot spends in any given area of the state space is proportional to the integral of the distribution over that same area (ergodicity cost) weighted against the effort to perform that trajectory (effort cost). Thus, a locally optimal trajectory for the robot has a cost function that involves two parts: the ergodicity cost and the effort cost.

---

**Algorithm 1** Ergodic Exploration of Entropy (E<sup>3</sup>)

---

- 1: Define  $t_D$  finite receding horizon time duration
  - 2: Define  $\tau$  confidence decay time constant
  - 3: Define  $l$  number of waypoints in  $u_i^0(t)$
  - 4: Define the  $Q$ 's,  $R$ 's,  $S$ 's optimization weighting matrices
  - 5: Init.  $t$  to 0
  - 6: Init.  $x(0)$  state of robot
  - 7: Init.  $\hat{\psi}_0(x)$  to uniform distribution
  - 8: Init.  $G_0(x)$  and  $I_0(x)$  to all zeros
  - 9: Init.  $i$  to 0
  - 10: **while**  $t < t_F$  **do**
  - 11:   Update time variables  $t_0 \leftarrow t$  and  $t_f \leftarrow t + t_D$
  - 12:   Update  $H_i(x) \leftarrow (1 - G_t(x))\hat{\psi}_t(x)$
  - 13:   Calc. initial  $u_i^0(t)$
  - 14:   Calc. optimal  $u_i^*(t)$  and  $x_i^*(t)$  using  $u_i^0(t)$  and  $H_i(x)$
  - 15:   Execute trajectory, updating  $\hat{\psi}_t(x)$  and  $G_t(x)$
  - 16:   Increment counter  $i \leftarrow i + 1$
  - 17: **end while**
- 

The ergodicity cost, as defined in [37] [34], is a norm on the differences in the Fourier coefficients of the state trajectory<sup>3,4</sup>,

$$F_k(x(T)) = \frac{1}{|T|} \int_T \overline{f_k(x(t))} dt, \quad (9)$$

and the Fourier coefficients of the spatial distribution,  $\phi(x)$ ,

$$\Phi_k = \int_X \overline{f_k(x)} \phi(x) dx. \quad (10)$$

In (9) and (10),  $f_k(x)$  represents the  $k$ th Fourier basis function,

$$f_k(x) = e^{2\pi j \xi_k^T x}, \quad (11)$$

where  $\xi_k \in \mathbb{R}^n$  is the  $k$ th spatial frequency (*a.k.a* wavenumber). The norm on the differences in Fourier coefficients is weighted more heavily on lower spatial frequencies with the Sobolev space norm,

$$\Lambda_k = \frac{1}{(1 + k^T k)^{\frac{(n+1)}{2}}}. \quad (12)$$

---

<sup>3</sup> $\overline{f_k(x)}$  represents the complex conjugate of the function  $f_k(x)$ .

<sup>4</sup>Notation notes:  $x(t)$  represents the state location at time  $t$  (i.e  $x(t) \in X$ ), where  $x(T)$  represents the state spatial trajectory over all  $T = [t_0, t_f]$ , (i.e.  $x(T) \in \mathcal{X} \subset \mathbb{L}_2^n[T]$ ).

The ergodicity cost is thereby defined as

$$J_{ergo}(x(T)) = \frac{1}{2}Q_e \sum_{k \in K} \Lambda_k \|F_k(x(T)) - \Phi_k\|, \quad (13)$$

where  $Q_e \geq 0$  is weighting value on the ergodicity cost,  $K = \{0, 1, \dots, N_1 - 1\} \times \dots \times \{0, 1, \dots, N_n - 1\}$  is the set of Fourier coefficient indices and  $N \in \mathbb{N}^n$  is the number of Fourier coefficients in each dimension of the state space<sup>5</sup>.

The effort cost is a weighted  $\mathbb{L}_2^n[T]$ <sup>6</sup> norm on the input trajectory,

$$J_{effort}(u(t)) = \frac{1}{2|T|} \int_T u(t)^\top R_e u(t) dt, \quad (14)$$

where  $R_e = R_e^\top \succ 0$  is the weighting matrix on the input.

Therefore, the total cost is

$$J(x(t), u(t)) = J_{ergo}(x(t)) + J_{effort}(u(t)), \quad (15)$$

and the locally optimal ergodic state trajectory,  $x^*(t)$ , and input,  $u^*(t)$ , are ones that minimizes  $J$  and satisfy the dynamics of the robot in (1). Thus, as in [37], the locally optimal state and input trajectories are

$$(x^*(t), u^*(t)) = \arg \min_{x(t), u(t)} J(x(t), u(t)). \quad (16)$$

Solving (16) is done iteratively using the projection-based trajectory optimization method [37] [17]. In each iteration two Linear Quadratic Regulator (LQR) problems are solved. The first LQR problem finds a non-feasible trajectory pair that reduces the cost in (15) and the second LQR problem projects this non-feasible trajectory pair back onto the manifold of feasible trajectory pairs. This iteration of solving LQR problems is continued until the cost can not be reduced any further.

Each one of the LQR problems has a trio of weighting matrices that weight the state, input, and final state cost. These weighting matrices are commonly denoted as  $Q$ ,  $R$ , and

---

<sup>5</sup>The Fourier basis functions and Fourier coefficients indices used in this chapter differ slightly from those used in [37] and [34].

<sup>6</sup> $\mathbb{L}_p^n[D]$  represents the space of Lebesgue integrable functions of  $n$ th-dimension with  $p$ th-power over the domain  $D$ .



$S$ , respectively. In the  $E^3$  algorithm these matrices are defined as  $Q_d$ ,  $R_d$ , and  $S_d$  for the first LQR problem that solves for the descent direction and  $Q_p$ ,  $R_p$ , and  $S_p$  for the second LQR problem that projects back onto the feasible trajectory manifold. Further details on the solving the minimization problem of (16) is provided in more detail in Section 2.5.

The distribution used in the ergodic trajectory optimization,  $\phi(x)$ , are different in the  $E^3$  algorithm and the EEDI algorithm. The  $E^3$  algorithm sets  $\phi(x)$  to a representation of information gain across the space encoded with the function  $H(x)$ , which uses the predicted information distribution,  $\hat{\psi}(x)$ , and the confidence of the predicted information distribution,  $G(x)$ . The EEDI algorithm sets  $\phi(x)$  to a map of expected value of the Fisher information of the measurement model with respect to the belief value of the parameters, which is called the Expected Information Density (EID).

### 2.3.2 Initializing Input Trajectory for Ergodic Optimization

The initial input trajectory has a large influence on the locally optimal trajectory returned by the ergodic optimization, as demonstrated empirically in this work as well as in the original work [37]. The impact of the initial trajectory is due to the algorithm’s inability to find a globally optimal solution, but instead it finds a locally optimal solution based on this initial trajectory. In addition, if the same initial trajectory is provided to the ergodic optimization algorithm over each iteration of the finite receding horizon control algorithm (e.g. EEDI or  $E^3$ ), the locally optimal trajectory will likely converge to the same trajectory over and over again. This convergence does not produce desirable results for on going exploration.

In the  $E^3$  algorithm the initial input trajectory,  $u_i^0(t)$ , is calculated based on the current information gain map,  $H_i(x)$ . The information gain map is treated like an elevation contour map or an intensity grayscale image. An image processing algorithm known as the *Watershed* algorithm [64] is used to find the highest  $l$  peaks in the information gain map. The shortest path between the robot’s current location and the  $l$  peaks is then calculated. Finding this type of shortest path is known as the “Traveling Salesman Problem” [46] and, in general, is

a very hard problem to solve. When  $l$  is small (say less than 10) the shortest path can be solved quickly using a dynamic programming technique [33]. A small  $l$  works well for the  $E^3$  algorithm, so solving for the shortest path becomes straight forward. The initial input trajectory,  $u_i^0(t)$ , is set to the input that gives the state trajectory that follows the shortest path between the “peaks” in the information gain map. The projection operation discussed in Section 2.3.1 is used to find an input trajectory given a state trajectory. The method used to generate the initial trajectory in the EEDI algorithm is not discussed in the various publications.

### 2.3.3 Updating Estimate of Information Distribution

As the robot executes the trajectory produced by the ergodic optimization algorithm, it collects samples from the underlying information distribution,  $\psi_t(x)$ , at each sample time. Each measurement collected updates the robot’s estimate of the information distribution,  $\hat{\psi}_t(x)$ , following the update law in (4).

The update law in the  $E^3$  algorithm uses the fact that the total information integrates to one; thus, the amount of information yet to be sampled is one minus the total amount of information already sampled. The total amount of information yet to be sampled is

$$\Gamma(\hat{\psi}, I) = 1 - \int_X \hat{\psi}(x)I(x)dx. \quad (17)$$

The update law distributes the total amount of information yet to be sampled, to all states that have yet to be sampled, proportionally to the state’s geodesic distance away from other states that have already been sampled and from the boundary of the search area. This geodesic distance of a state  $x$  is defined with the function  $L(x, I)$ . If the indicator function,  $I$ , is treated as a binary image, the geodesic distance from any state that has been sampled can be quickly calculated with an image processing algorithm known as the *Fast Marching Method* [49]. Fig. 2 shows a 2-dimensional example of the geodesic distances (normalized by  $\frac{|X|}{C}$ ) of the non-sampled states (shaded areas) from the boundary of the search area (black areas) and from others states that have already been sampled (white areas) for a given robot

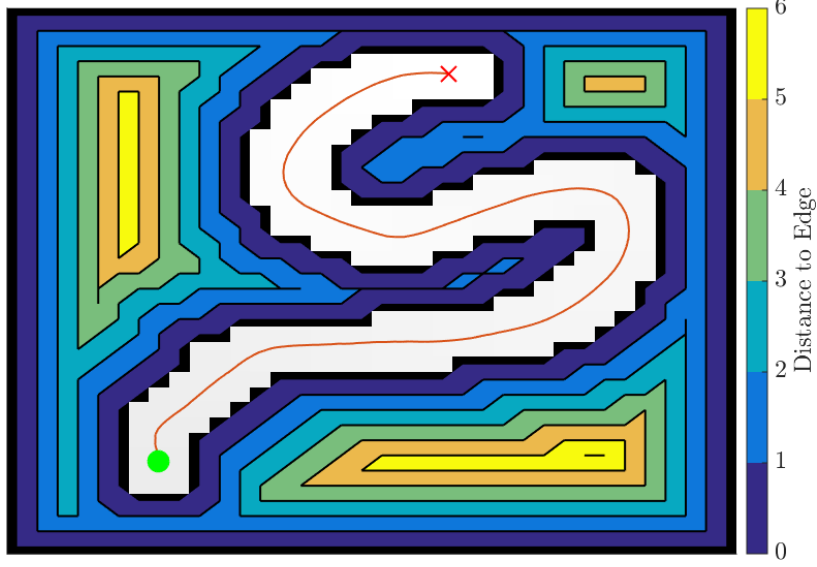


Figure 2: Geodesic distances (normalized by  $\frac{|X|}{C}$ ) for non-sampled states (shaded areas) from the search area boundaries (black areas) and from other sampled states (white areas) for a given robot trajectory (solid line).

trajectory (solid line).

Thus, the update function is defined as

$$\Upsilon(x, \hat{\psi}, I) = \frac{\Gamma(\hat{\psi}, I) L(x, I)}{\int_X L(x, I) dx}. \quad (18)$$

Assuming that the distribution integrates to one is a big assumption. This assumption allows for a natural way to estimate the underlying distribution for areas that have never been sampled. In most cases it is reasonable to assume what is the total amount of information in the given space; thus, the distribution can always be normalized to unity. However, if the total amount of information is not known or can not be reasonably estimated then this could cause problems for the  $E^3$  algorithm. If the total amount of information is under estimated then the robot can get stuck in one region thinking it has sufficiently explored the space. If the total amount of information is over estimated then the robot can expend too much effort exploring areas that have already been sufficiently explored.

### 2.3.4 Additional Notes on $E^3$

It is worth noting a few aspects of the Fourier transform that may cause problems for the ergodic optimization (Section 2.3.1) that have not been explicitly stated in previous work on this subject. The ergodic optimization as it is formulated in [38] assumes that the distribution,  $\phi(x)$ , is a *continuous* function across the state space with finite support. Therefore, because of the bounds on  $X$ , the continuous Fourier transform of the distribution produces discrete values (Fourier series coefficients) that are used in calculating the ergodic cost and, thus, the locally optimal trajectories. However, the inverse Fourier transform of these Fourier series coefficients produces an unbound periodic function with period defined by the bounds of  $X$ . This periodic function means that the locally optimal trajectory will not respect the bounds of  $X$  and will attempt to be ergodic for not only states *inside* of  $X$  but also states *outside* of  $X$ .

Often if the ergodic trajectory is initialized near the edge of the state space  $X$  it will get stuck on the boundary while trying to optimize for areas of the state space that it can not reach. To fix this problem, the distribution given to the ergodic optimization can be “padded” with zeros. This zero padding spreads out the bounds on  $X$ ; thus, focusing the optimization of the ergodic trajectory more heavily on information inside  $X$  verses outside of  $X$ .

Additionally, it is worth noting that if the estimate of the information distribution,  $\hat{\phi}_t(x)$ , is a *discrete* function across the state space,  $X$ , then some additional precautions must be taken while performing the ergodic optimization. In calculating the ergodic cost, the discrete Fourier transform is required to calculate the Fourier coefficients of the discrete distribution. Using the discrete Fourier transform will produce periodic discrete Fourier coefficients. These Fourier coefficients will have a period equal to the inverse of the sampling spacing of the discrete distribution in  $X$ . If the distribution has frequency information that is more than half the sampling frequency (Nyquist frequency) then that information will be aliased to the lower frequencies in the Fourier coefficients. Since the lower frequency are predominately

used in weighting variable  $\Lambda_k$ , this aliasing will produce a non-optimal ergodic trajectory. This can be fixed by increasing the sampling frequency of the information distribution in the state space.

## 2.4 *Exploration vs. Exploitation*

The  $E^3$  algorithm eloquently deals with the common problem of when to explore for new information and when to exploit the current information to get the best desired performance [23]. The  $E^3$  algorithm does not explicitly decide when to explore or when to exploit, but instead it is always exploiting the current estimate of the underlying information distribution. However, while exploiting the current information, the robot will naturally explore for new information, due to fact that the trajectory is ergodic with respect to the given distribution.

There are number of methods for dealing with exploration and exploitation [67]. These methods fall into two main categories: *undirected exploration* and *directed exploration* [58]. Undirected exploration does not utilize exploration-specific knowledge and explores using randomness in the decision process [42]. The simplest example of undirected exploration is the *random walk* method, where every decision is decided on randomly with some chosen distribution (usually uniform) over the possible choices. Directed exploration uses knowledge of the learning process itself and directs exploration to maximize knowledge gain. An example of directed exploration is the *counter-based exploration* method, where decisions are made based on how often states have occurred due to the past decisions that have been made [48].

The  $E^3$  algorithm actually combines characteristics from both the undirected and directed methods of exploration. There is a undirected method of exploration known as *utility-driven probability distributions*, where a decision is made based on the utility of gaining new information. This type of exploration has the characteristic that it explores and exploits simultaneously [58]. There is a a directed method of exploration known as *counter-based exploration with decay*, which is similar to *counter-based exploration* but the state counts decay with some exponential rate over time. This method will favor exploration decisions

which visit states that have not been recently encountered. This type of exploration has shown to perform well in dynamic state spaces [59].

Looking at (6) one can see why the  $E^3$  algorithm has characteristics of both the *utility-driven probability distributions* method and the *counter-based exploration with decay* method. The information content,  $\psi(x)$ , is the utility function needed for the *utility-driven probability distributions* method and the uncertainty,  $(1 - G(x))$ , is essentially an exponentially decaying counter for the states that have been visited, which is needed for the *counter-based exploration with decay* method. Thus, the information gain function,  $H(x)$ , is what allows the  $E^3$  algorithm to eloquently and effortlessly deal with the exploration verse exploitation conundrum.

## 2.5 Minimizing Ergodicity and Effort

An objective, as stated in Section 2.2, is to minimize ergodicity and effort in order to preform optimal coverage. To perform this minimization it is necessary to find an optimal feasible state and input trajectory tuple,  $z^*(T)$ , which minimizes the cost  $J(z(T))$ ,

$$z^*(T) = \arg \min_{z \in \mathcal{Z}} J(z(T)), \quad (19)$$

where  $z(T) = (x(T), u(T))$  represents a feasible state and input trajectory tuple.

An iterative approach is used to solve optimization problem in (19). At each iteration of the optimization algorithm, a descent direction,  $\zeta(T)$ , in the trajectory tuple,  $z(T)$ , is calculated. The solution is found by moving along this descent direction. The descent direction is the one that minimizes the quadratic model of the form <sup>7,8</sup>

$$\zeta^*(T) = \arg \min_{\zeta(T) \in T_{\sigma(T)}\Sigma} D_{\sigma(T)}J(z(T)) \circ \zeta(T) + \frac{1}{2} \|\zeta(T)\|_{Q_d, R_d, S_d}^2. \quad (20)$$

Intuitively, the first term of (20) picks the decent direction that moves from the current feasible trajectory tuple,  $z(T)$ , to the local optimal infeasible trajectory tuple. The second

---

<sup>7</sup> $\circ$  represents a general linear operator.

<sup>8</sup> $D_x J(x)$  represents the derivative of  $J$  with respect to  $x$ . The subscript will be left off when there is only one argument in the function.

term of (20) limits the norm on the on decent direction so that multiple steps are required and prevents instability in the iterations of the algorithm. A cartoon representation of this optimization is shown in Fig. 3.

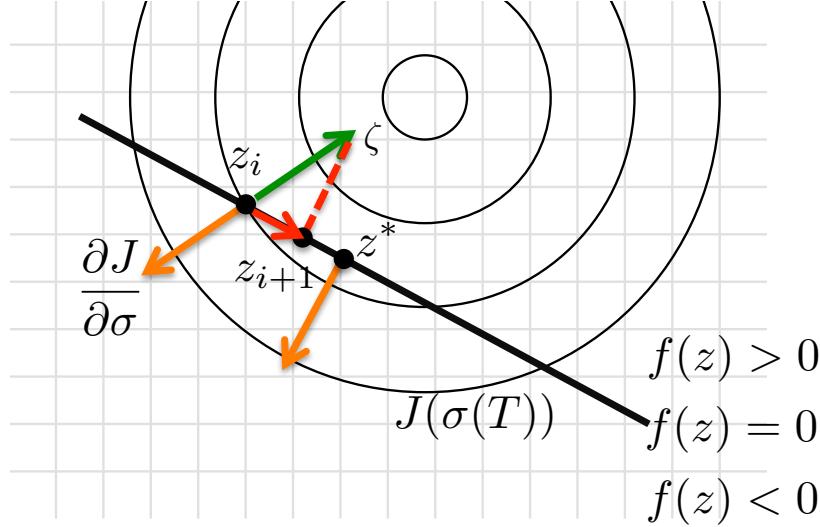


Figure 3: Cartoon intuitive representation of how the projection optimization works. Starting at a feasible trajectory (all feasible trajectories line on the thick black line)  $z_i$ , the decent direction (green arrow),  $\zeta$ , that moves the the trajectory to a minimal cost (contours of cost are shown in thin black lines), which is an infeasible trajectory. The projection operator projects (red dash line) the infeasible trajectory back to the feasible manifold,  $z_{i+1}$ . This is repeated until the trajectory reaches the local feasible optimal trajectory at  $z^*$ .

The trajectory tuple is updated by moving in the descent direction with some step size  $\gamma$ . This in general will result an infeasible trajectory tuple,  $\sigma(T) = z(T) + \gamma\zeta(T)$ . An infeasible trajectory tuple,  $\sigma(T) = (\chi(T), \mu(T)) \in \Sigma$ , is one where the an arbitrary input trajectory,  $\mu(T) \in \mathcal{U}$ , to the dynamical system does result in the state trajectory,  $\chi(T) \in \mathcal{X}$ . This infeasible trajectory tuple is projected back onto the feasible manifold with the projection operator  $P$ ,

$$z_{i+1}(T) = P(\sigma_i(T)) = P(z_i(T) + \gamma_i\zeta_i(T)). \quad (21)$$

This continues until  $\|\mathbf{D}_{\sigma(T)}J(z_i(T))\| < \epsilon$ .

Both (20) and (21) are solved by formulating them into Linear Quadratic (LQ) optimal control problems. These LQ problems are solved in the standard way of integrating the Riccati equation backwards in time

### 2.5.1 Descent Direction Derivation

The descent direction minimization problem of (20) can be rewritten into a quadratic form with the first term written as,<sup>9</sup>

$$\begin{aligned} \mathbf{D}J(\sigma(T)) \circ \zeta(T) &= \mathbf{D}_\chi \left( \frac{1}{2} \sum_{k \in K} \Lambda_k \|E_k(\chi(T))\|^2 \right) \circ \eta(T) + \mathbf{D}_\mu \left( \frac{1}{2} \|\mu(T)\|_{R_e}^2 \right) \circ \nu(T) \\ &= \int_T q_d(\chi(t))^\top \eta(t) + r_d(\chi(t))^\top \nu(t) dt, \end{aligned} \quad (22)$$

where

$$\begin{aligned} q_d(\chi(T)) &\in \mathbb{L}_2^n[T] \text{ and } q_d(\chi(T)) = \frac{-2\pi i}{|T|} \sum_{k \in K} \Lambda_k E_k(\chi(T))(f_k(\chi(T)) \circ \xi_k) \\ r_d(\mu(T)) &\in \mathbb{L}_2^m[T] \text{ and } r_d(\mu(T)) = R_e \circ \mu(T). \end{aligned}$$

Next, the second term of (20) can be rewritten as

$$\frac{1}{2} \|\zeta(T)\|_{Q_d, R_d, S_d}^2 = \frac{1}{2} \int_T \eta(t)^\top Q_d \eta(t) + \nu(t)^\top R_d \nu(t) dt + \frac{1}{2} \eta(t_f)^\top S_d \eta(t_f), \quad (23)$$

where  $Q_d = Q_d^\top \succeq 0$ ,  $R_d = R_d^\top \succ 0$ , and  $S_d = S_d^\top \succeq 0$  are arbitrary weighting matrices on the quadratic terms for the descent direction. Substituting (22) and (23) into (20) produces

$$\begin{aligned} \zeta(T)^* &= \arg \min_{\zeta(T) \in \mathbf{T}_{\sigma(T)}\Sigma} \mathbf{D}J(\sigma(T)) \circ \zeta(T) + \frac{1}{2} \|\zeta(T)\|_{Q_d, R_d, S_d}^2 \\ &= \int_T \frac{1}{2} \eta(t)^\top Q_d \eta(t) + \frac{1}{2} \nu(t)^\top R_d \nu(t) + q_d(x(t))^\top \eta(t) + r_d(u(t))^\top \nu(t) dt + \frac{1}{2} \eta(t_f)^\top S_d \eta(t_f), \end{aligned} \quad (24)$$

which is quadratic in  $\zeta(T)$ .

---

<sup>9</sup> $\top$  is overloaded to represent conjugate transpose and the adjoint.



Since  $\zeta(T) = (\eta(T), \nu(T))$  must lie in the tangent space of the trajectory manifold (i.e.  $\zeta(T) \in \mathbb{T}_{\sigma(T)}\Sigma$ ), it must satisfy the linearized system dynamics. This property means

$$\dot{\eta}(t) = A(t)\eta(t) + B(t)\nu(t), \quad A(t) = \frac{\partial f(\chi(t), \mu(t))}{\partial x}, \quad B(t) = \frac{\partial f(\chi(t), \mu(t))}{\partial u}.$$

Thus, solving for  $\zeta(T)^* = (\eta(T), \nu(T))^*$  is equivalent to solving a generalized LQ problem, and is solved by integrating the following differential equations backwards in time,

$$\dot{P}_d(t) = -P_d(t)A(t) - A(t)^\top P_d(t) - Q_d + P_d(t)B(t)R_d^{-1}B(t)^\top P_d(t),$$

$$P_d(t_f) = S_d,$$

$$\dot{W}_d(t) = (P_d(t)B(t)R_d^{-1}B(t)^\top - A(t)^\top)W_d(t) - q_d(\chi(t)) - P_d(t)B(t)R_d^{-1}r_d(\chi(t)),$$

$$W_d(t_f) = 0.$$

$$\nu(t) = -R_d^{-1}B(t)^\top P_d\eta(t) - R_d^{-1}(B(t)^\top W_d + r_d(\chi(t))),$$

$$\dot{\eta}(t) = A(t)\eta(t) + B(t)\nu(t).$$

## 2.6 Feasible Trajectory Projection

The  $\eta(T)$  and  $\nu(T)$  found by solving the LQ problem in Section 2.5.1 form the decent direction tuple,  $\zeta(T)$ , which in general will not point to a feasible trajectory tuple,  $z(T) = (x(T), u(T))$ , but instead to an infeasible trajectory tuple,  $\sigma(T) = (\chi(T), \mu(T))$ . A projection operation is used to project the infeasible trajectory onto the feasible trajectory manifold,

$$z(T) = P(\sigma(T)),$$

where

$$P(\sigma(T)) : \begin{cases} u(t) = \mu(t) + K_p(t)(\chi(t) - x(t)) \\ \dot{x}(t) = f(x(t), u(t)) \end{cases}.$$

The optimal feedback gain  $K_p(t)$  is found by solving an additional LQ problem,

$$\dot{P}_p(t) = -P_p(t)A(t) - A(t)^\top P_p(t) - Q_p + P_p(t)B(t)R_p^{-1}B(t)^\top P_p(t), \quad P_p(t_f) = S_p,$$

$$K_p(t) = -R_p^{-1}B(t)^\top P_p(t),$$

where  $Q_p = Q_p^\top \succeq 0$ ,  $R_p = R_p^\top \succ 0$ , and  $S_p = S_p^\top \succeq 0$  are weighting matrices.

Interesting results occur when adjusting the weighting matrices for the projector operator. An intuitive explanation of each weighting matrix is as follows:

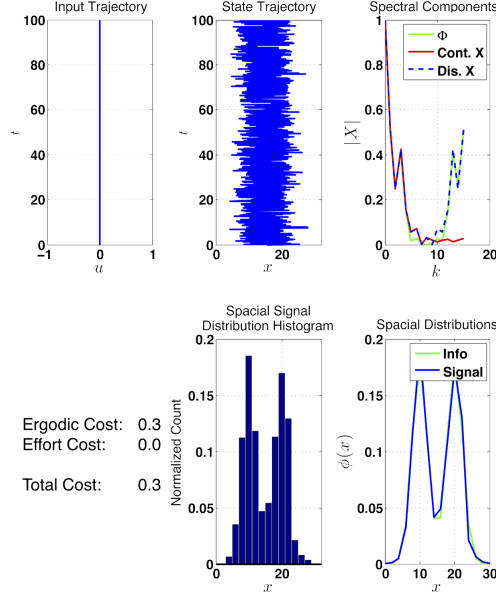
$Q_p$ : Sets difficulty in moving the state trajectory from the initial infeasible state trajectory.

$R_p$ : Sets difficulty in moving the input trajectory from the initial infeasible input trajectory.

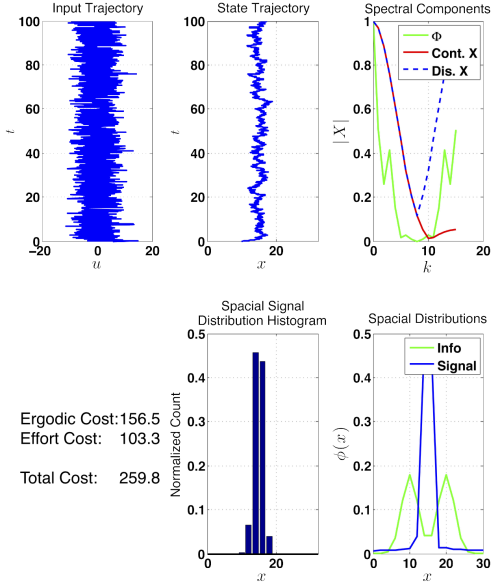
$S_p$ : Sets importance for the final state value to be close to the given final state value.

The plots in Fig. 4 give insight into how the weighting matrices affect the projection from an infeasible input and state trajectory tuple to a feasible tuple. The plots show the results doing the trajectory projection for a simple system with single integrator dynamics. In the top left mini-plot of Fig. 4 the optimal input trajectory (input equals to zero for all time) and the optimal state trajectory are shown (state trajectory distribution equals desired distribution). However, this input and state trajectory tuple is infeasible (i.e. does not satisfy the system dynamics). The goal is to project the infeasible trajectory tuple to a feasible feasible trajectory tuple with the an input trajectory (top-left mini-plots) as close to as possible to the optimal input trajectory (zero for all time) and the state trajectory spatial distribution (bottom-right mini-plots) (or histogram (bottom-middle mini-plots)) as close as possible to the the desired distribution. The cost on how close the state trajectory spatial distribution is to the desired distribution is measured in the transformed space (top-right mini-plots).

Two different weighting matrix combinations are used to project the infeasible input and state trajectory tuple to a feasible tuple, these results are shown in Fig. 4b and Fig. 4c. As can be seen in the figures, one must chose these weighting matrices carefully. A much higher cost on effort must be tolerated to allow for a decent fit in the state trajectory distribution, which can be seen in the bottom right mini-plots of Fig. 4b and Fig. 4c.

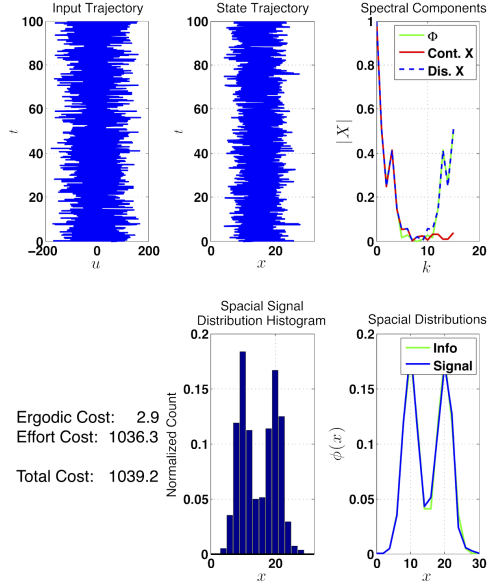


(a) Initial infeasible optimal trajectories (input equal to zero, state distribution equal to desired distribution) before being applied to projection operator.



(b) Feasible trajectories.

$$R_p = 1 \text{ and } S_p = 1$$



(c) Feasible trajectories.

$$R_p = 0.01 \text{ and } S_p = 0.01$$

Figure 4: Results from applying projection operator to simple single integrator system. Plotted is the input (top left), state (top middle), and spatial transform components (top right), histogram of the state trajectory (bottom right), and the desired and actual information distribution (bottom right).

---

**Algorithm 2** Armijo Step Size Calculation Algorithm

---

```
1: function ARMIJO( $\alpha, \beta, J, P, z(T), \zeta(T)$ )
2:    $\kappa \leftarrow 0$ 
3:    $J_0 \leftarrow J(\sigma(T))$ 
4:    $J_1 \leftarrow \infty$ 
5:   while  $J_1 - J_0 > -\alpha\beta^\kappa \|\mathbf{D}_{u(T)}J(z(T))\|$  do
6:      $\gamma \leftarrow \beta^\kappa$ 
7:      $\chi(T) \leftarrow x(T) - \gamma\eta(T)$ 
8:      $\mu(T) \leftarrow u(T) - \gamma\nu(T)$ 
9:      $z(T) \leftarrow P(\sigma(T))$ 
10:     $J_1 \leftarrow J(z(T))$ 
11:     $\kappa \leftarrow \kappa + 1$ 
12:   end while
13: end function
```

---

## 2.7 Armijo Step Size Calculation

The descent direction step size,  $\gamma$ , is calculated using an Armijo line search algorithm [2]. The algorithm is shown in Algorithm 2.

For intuition into the Armijo parameters, one can think about  $\alpha$  as determining how much one cares about the improvement of the cost. If  $\alpha$  is large the improvement will be a larger fraction of the norm of the signal and the convergence will be slower. However, if  $\alpha$  is too small several iteration will need to be performed to reach satisfactory convergence results.

One can think of  $\beta$  as determining how course of a search takes place. If  $\beta$  is large the search will be very fine and the convergence will be slower. However, if  $\beta$  is too small the search may be too course and the optimal will not be found.

## 2.8 Conclusion

This chapter introduced the *Ergodic Environmental Exploration* ( $E^3$ ) algorithm, which optimally explores unknown environments with areas of varying degrees of importance. The  $E^3$  algorithm generates a trajectory for a dynamical system to discover and collect information from an unknown underlying information distribution in the space of interest. The algorithm minimizes control effort while also minimizing the ergodic cost of the trajectory; the

difference between the time average behavior of the trajectory to the spatial distribution of maximum information gain. By using the of the information gain as the distribution for the generation of the optimal ergodic trajectory, the algorithm effectively and easily deals with the issue of when to explore and when to exploit.

This chapter presented the theory underlying the  $E^3$  algorithm, or the “hammer”, used to optimally guide an agent through the exploration of an unknown space. The next chapter introduces the “nail” to the  $E^3$  algorithm. Experiments are discussed involving real robots that run the  $E^3$  algorithm to explore a space in the ground and in the air.

## CHAPTER III

### LEARNING OPTIMAL COVERAGE (EXPERIMENTS)

#### ***3.1 Experimental Setup Introduction***

The  $E^3$  algorithm outlined in Section 2.3 was experimentally tested in simulation and validated on real robots. The first robot that was used for the experiment was the Khepera 3, a differential drive robot that is approximately 13 cm in diameter (see Fig. 5a). The second robot that was used for the experiment was a quadrotor, which is approximately 25 cm in diameter (see Fig. 5b).

Experiments were conducted in a lab equipped with a 10 camera OptiTrack<sup>®</sup> motion capture system and a work station to run control computations and for communication with the robots. The reflective balls mounted on the robots in Fig. 5 are used by the motion capture cameras to identify the robots and compute their position and orientation.

#### ***3.2 Differential Drive Robot***

Differential drive robots are ground robots that consist of two independently powered wheels that are non-holonomic (i.e. contain velocity constraints); the wheels can not slide perpendicular to the rolling direction. In addition, a differential drive robot often has a caster wheel or support ball that is holonomic (i.e. contains no velocity constraints); the caster wheel is free to move in any direction. The powered wheels are positioned parallel to one another and control the robots linear and angular velocities by varying the rotational angular velocity of the each wheel.

##### **3.2.1 Differential Drive Dynamics**

The Khepera robot is a differential drive robot, which has states consisting of its position in the plane,  $(p_1, p_2)$ , and its heading angle,  $\theta$ . Thus, the state vector of a differential drive robot



(a) Khepera 3 robot with motion capture reflector ball hat.



(b) Quadrotor robot with motion capture reflector ball hat.

Figure 5: Robots used for  $E^3$  experiments

is  $x = [p_1, p_2, \theta]^T$ . The inputs to the robot are the left and right motor values,  $(w_L, w_R)$ , which directly control wheel angular velocities. In defining the dynamics it is easier to use the robot's linear velocity,  $v$ , and angular velocity,  $\omega$ , as the inputs to the robot; thus, the input vector is  $u = [v, \omega]^T$ . Using the linear and angular velocity in the input is possible because there is a one-to-one mapping between  $[v, \omega]^T$  and  $[w_L, w_R]^T$  for differential drive robots, via the function

$$\begin{bmatrix} w_L \\ w_R \end{bmatrix} = g(u) = \begin{bmatrix} K_C(v - \omega K_B)/K_R \\ K_C(v + \omega K_B)/K_R \end{bmatrix} \quad (25)$$

where  $K_B$ ,  $K_R$ , and  $K_C$  are the robot's wheel base, wheel radius, and wheel angular velocity to motor value constant; respectively. For the Khepera 3 robot the parameters are shown in Table 1.

A differential drive robot has dynamics that can be model with the, so called, unicycle

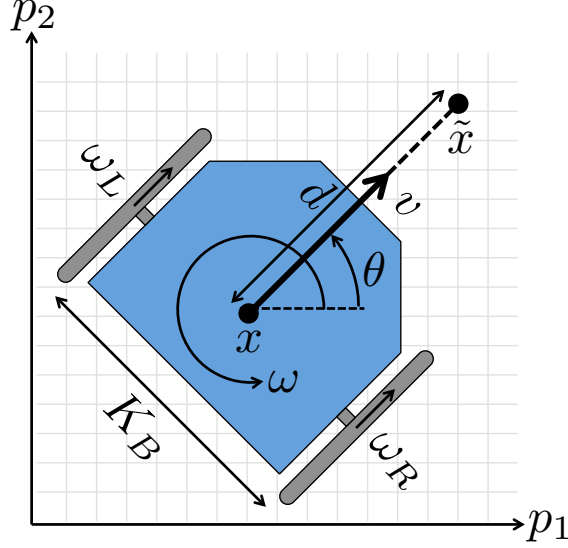


Figure 6: Differential drive robot cartoon. The positional portion of the state is at  $x$  while the diffeomorphic positional portion of the state is at  $\tilde{x}$ .

dynamics,

$$\dot{x} = f(x, u) = \begin{bmatrix} v \cos(\theta) \\ v \sin(\theta) \\ \omega \end{bmatrix}. \quad (26)$$

### 3.2.2 Differential Drive Control

The dynamics of a differential robot (e.g Khepera robot), as seen in (26), are non-linear and non-holonomic. The non-linear dynamics are due to the trigonometric functions of the heading angle. The non-holonomic dynamics are due to the velocity constraint in that robot can not instantaneously move perpendicular to its heading angle. Control design for a non-linear and non-holonomic system is more difficult.

There are several strategies for controlling differential drive robots [10]. The strategy employed in this work is to not control the true state of the system, but instead control a diffeomorphic state that has holonomic dynamics. The holonomic state dynamics are linearized at each instance in time and a linear quadratic regulator (LQR) is then used to control the robot.



### 3.2.2.1 Differential Drive Diffeomorphic State Dynamics

The main idea for a differential drive robot's diffeomorphic state is to project a state out in front of the robot by some distance  $d$ , and to control this state instead of the true state of the robot. One can imagine that this state sits at the end of a rigid rod of length  $d$  that is connected to the front of the robot. The end of this rod has no velocity constraints because it can move both forward and backward as well as left and right instantaneously, hence it is holonomic. Since this new state is diffeomorphic to the true state, any motion from the diffeomorphic state can be directly mapped to motion in the true state. One other caveat in controlling a diffeomorphic state instead of the true state is that the diffeomorphism has to be applied to the desired state as well.

We can derive the new diffeomorphic state,  $\tilde{x}$ , and diffeomorphic state dynamics using the true state,  $x$ ; offset,  $d$ ; and the true state dynamics from (26). Thus, the diffeomorphic state is

$$\tilde{x} = \begin{bmatrix} p_1 + d \cos \theta \\ p_2 + d \sin \theta \\ \theta \end{bmatrix}. \quad (27)$$

Taking the time derivative of (27) we get the diffeomorphic state dynamics

$$\dot{\tilde{x}} = \tilde{f}(x, u) = \begin{bmatrix} \dot{p}_1 - d \sin \theta \dot{\theta} \\ \dot{p}_2 + d \cos \theta \dot{\theta} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cos \theta - d \omega \sin \theta \\ v \sin \theta + d \omega \cos \theta \\ \omega \end{bmatrix}. \quad (28)$$

To formulate the linear system we linearizing the dynamics in (28) around the current state

and input

$$\left. \frac{\partial \tilde{f}}{\partial x} \right|_{\theta, v, \omega} = \begin{bmatrix} 0 & 0 & -v \sin \theta - d\omega \cos \theta \\ 0 & 0 & v \cos \theta - d\omega \sin \theta \\ 0 & 0 & 0 \end{bmatrix}, \quad (29)$$

$$\left. \frac{\partial \tilde{f}}{\partial u} \right|_{\theta, v, \omega} = \begin{bmatrix} \cos \theta & -d \sin \theta \\ \sin \theta & d \cos \theta \\ 0 & 1 \end{bmatrix}. \quad (30)$$

The above linear system is not completely controllable for  $\tilde{x}$ . This lack of controllability here is sufficient because we are only interested in controlling the robot's diffeomorphic position in the plane, not its orientation. Thus, we can ignore the last column and row in (29) and the last row in (30) to get our linearized diffeomorphic state dynamics

$$\dot{\tilde{x}} = A\tilde{x} + Bu, \quad (31)$$

where

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad B(\theta; d) = \begin{bmatrix} \cos \theta & -d \sin \theta \\ \sin \theta & d \cos \theta \end{bmatrix}. \quad (32)$$

It makes sense that the  $A$  matrix equals zero because all differential drive robots are drift-free systems (i.e. state time derivative equals zero with zero input).

### 3.2.2.2 Differential Drive Position Controller

Using the linearized diffeomorphic state dynamics derived in (31), a state feedback linear quadratic regulator is used to drive the differential drive robot from its current state,  $x$ , to any other state,  $\bar{x} \in \mathbb{R}^2$ , in the plane. The current state will determine a diffeomorphic

Table 1: Parameters used to control the Khepera differential drive robot.

Variable	Value
$K_B$	0.0885 m
$K_R$	0.021 m
$K_C$	3335.8
$d$	0.01 m
$Q$	diag(10,10)
$R$	diag(20,1)

position and the desired state will determine a desired diffeomorphic position,

$$\tilde{p} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix}, \quad (33)$$

$$\tilde{\tilde{p}} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} + d \begin{bmatrix} \cos(\bar{\theta}) \\ \sin(\bar{\theta}) \end{bmatrix}. \quad (34)$$

The diffeomorphic position and the desired diffeomorphic position determine the state-feedback control,  $u = -Ke = -K(\tilde{p} - \tilde{\tilde{p}})$ , where  $K$  is the LQR gain value that minimizes the quadratic cost

$$J(u) = \int_0^\infty e^\top Q e + u^\top R u \, dt, \quad (35)$$

subject to the dynamics in (31). For this work the parameters used to control the Khepera are listed in Table 1.

### 3.2.3 Differential Drive Robot Experimental Setup

In this study, the robot was given the task of exploring a 2-dimensional area with a simulated underlying information distribution. The approximate size of the area to explore is 5 m  $\times$  3.75 m (18.75 m<sup>2</sup>), which was partitioned into a 36  $\times$  36 grid<sup>1</sup>. Each cell of the grid contains a constant amount of the information distribution. The information distribution used in the experiment is shown in Fig. 9a. With a sensor delta disc footprint  $\delta$  value set to 0.1 m, the

---

<sup>1</sup>To efficiently calculate Fourier coefficients grids with  $2^p$  cells are used.

Table 2: Variable constant values used in the  $E^3$  algorithm robot exploration experiments.

Variable	Value	Variable	Value
$\delta$	0.1 m	$Q_e$	16
$\Delta t$	0.25 s	$R_e$	diag(20,1)
$t_f$	600 s	$Q_d$	diag(1,1,1)
$T$	40 s	$R_d$	diag(1,1)
$\tau$	200 s	$S_d$	1e-6 diag(1,1,1)
$l$	4	$Q_p$	diag(1,1,1)
$C$	36	$R_p$	diag(1,1)
$N$	[32, 32]	$S_p$	1e-6 diag(1,1,1)

robot is able to measure the information distribution in the cell it is currently in as well as each adjacent cell<sup>2</sup>.

The laboratory where the experiments took place is equipped with a motion capture system to estimate the robot’s state. A computer in the lab ran the  $E^3$  algorithm, maintained the information distribution values, and simulated the robot’s information distribution sensor. In addition, the computer collected the state information from the motion capture system and sent motor input commands to the robot in real-time via WiFi. To better visualize the exploration in the area of interest, a projector system, which is mounted on the ceiling of laboratory, was used to project the confidence function’s values onto the floor where the robot was operating. The robot was set to explore the area of interest for 10 minutes.

The heading state’s Fourier components are all set to zero in the ergodic optimization, since the area of exploration and information distribution are in two,  $(p_1, p_2)$ , of the three state components. Consequently, the robot does not attempt to be ergodic in its trajectory with respect to its heading information. The other constants values needed in the  $E^3$  algorithm that were used in the experiment are listed in Table 2.

---

<sup>2</sup>8-connected grid adjacency is used.

### 3.3 Quadrotor Robot Experimental Setup

In addition to the Khepera differential drive ground robot, the  $E^3$  algorithm was tested on a custom built quadrotor (see Fig. 5b), called the GRITS quad. The GRITS quad uses a custom designed 250 mm in diameter carbon fiber chassis. The mass of the quad with the chassis, motors, electronic speed controls, microcontroller, wireless transmitter, battery, and reflector ball hat is 425 grams. Making the GRITS quad as small as possible was necessary so that the quadrotor was able to be operated in the same laboratory space that was used for the Khepera, which contains the motion capture equipment, projector, and computer running the  $E^3$  algorithm.

The states and dynamics for the GRITS quad are more complex than that of the Khepera because it moves around in 3-dimensions in the air verses 2-dimensions on the ground. The full state of a quadrotor are made from its position,  $\xi = [x, y, z]^T$ ; its attitude,  $\eta = [\phi, \theta, \psi]^T$ , represented as Euler angles; its linear velocity,  $\dot{\xi} = [\dot{x}, \dot{y}, \dot{z}]^T$ ; and its angular velocity,  $\dot{\eta} = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ , represented as Euler angle rates. The generalized coordinates consist of its position and attitude,  $q = [\xi, \eta]^T \in SO(3)$ , and the full state is

$$x = \begin{bmatrix} \xi \\ \eta \\ \dot{\xi} \\ \dot{\eta} \end{bmatrix} \in SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3 \quad (36)$$

The force,  $F_1, F_2, F_3, F_4$ , and torque,  $T_1, T_2, T_3, T_4$ , generated by the four motors of a quadrotor are the input to the dynamical system. For modeling the dynamical system is it more convenient to use torques around the Euler angle rotational axes,  $\tau_\eta = [\tau_\phi, \tau_\theta, \tau_\psi]^T$ , and the total force parallel to the quadrotor's  $z$ -axis,  $F_z$ , as inputs to the dynamical system.

#### 3.3.1 Quadrotor Dynamics

Each quadrotor has intrinsic properties that are unique to each system: mass, moment of inertia, arm length, motor force constants, and motor torque constants. It is assumed that

the center of mass is at the origin of the quadrotor reference frame and that the principle axes align with the axes of this reference frame. It is assumed that the magnitude of force and torque generated by each motor is related to the the motor input, which is a pulse width modulated (PWM) signal, with the following relationship,

$$F_i = M_i^2 C_{F_i} > 0 \quad \forall i, \quad (37)$$

$$T_i = M_i^2 C_{T_i} > 0 \quad \forall i. \quad (38)$$

$$F = F_1 + F_2 + F_3 + F_4 > 0, \quad (39)$$

The quadrotor can be model as rigid body with the full state consisting of position, linear velocity, orientation, and angular velocity. The orientation (or attitude) of the quadrotor lives in the space  $SO(3)$ , and for almost all orientations can be expressed uniquely using Euler angles,

$$\eta = [\phi, \theta, \psi]^\top. \quad (40)$$

These are the roll, pitch, and yaw angles, respectively, expressed from the quadrotor nominal frame (see Fig. 7) to the current quadrotor frame. The linear and angular velocities are the time derivatives of the position and orientation of the quadrotor and are expressed as

$$\dot{\xi} = [\dot{x}, \dot{y}, \dot{z}]^\top, \quad (41)$$

$$\dot{\eta} = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^\top. \quad (42)$$

Thus, the generalized coordinates for the quadrotor are expressed as

$$q = [\xi, \eta]^\top. \quad (43)$$

It is important to note that for intuition and representation it is easier to use Euler angles (roll, pitch, yaw) and Euler angular rates (roll rate, pitch rate, yaw rate) to represent orientation and angular velocity. For derivation of the dynamics it necessary to use a rotation

matrix that expresses the quadrotor frame in the local inertial frame<sup>3</sup>,

$$R = {}_L R_Q \in SO(3), \quad (44)$$

and a angular velocity vector, which is the derivative of the quadrotor frame to the local inertial frame expressed in the quadrotor frame,

$$\Omega = {}_Q \omega_{LQ} \in \mathbb{R}^3. \quad (45)$$

The Euler angles used here are roll, pitch, and yaw angles. These are rotations around the current frame. Starting with rotating around the nominal quadrotor frame z-axis by  $\psi$  radians, then around the new y-axis by  $\theta$  radians, and finally around the new x-axis by  $\phi$  radians. Thus, the representation of the current frame in the nominal quadrotor frame and local fixed frame is

$$\overline{Q} R_Q = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix}, \quad (46)$$

$${}_L R_{\overline{Q}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad (47)$$

$${}_L R_Q = {}_L R_{\overline{Q}} \overline{Q} R_Q = \begin{bmatrix} c_\theta s_\psi & c_\phi c_\psi + s_\phi s_\psi s_\theta & c_\phi s_\psi s_\theta - c_\psi s_\phi \\ c_\psi c_\theta & c_\psi s_\phi s_\theta - c_\phi s_\psi & s_\phi s_\psi + c_\phi c_\psi s_\theta \\ s_\theta & -c_\theta s_\phi & -c_\phi c_\theta \end{bmatrix}. \quad (48)$$

---

<sup>3</sup>Here the left subscript signifies the frame the quantity is expressed in, or projected onto. The right subscript is a label or the name of the frame or vector. For velocity the right subscript signifies the which frame the derivative is taken with respect to.

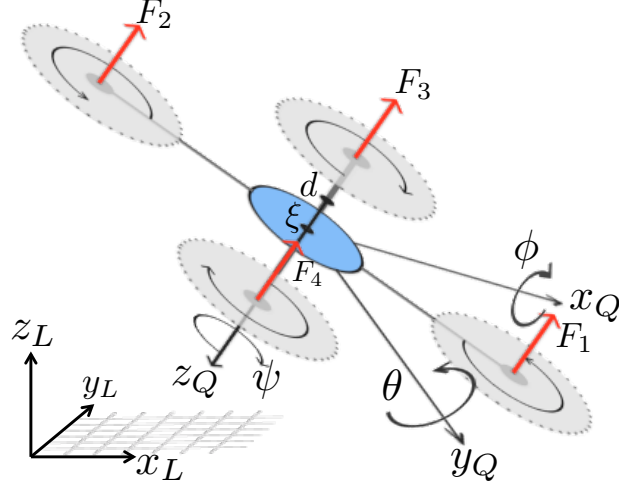


Figure 7: Quadrotor robot cartoon. The positional portion of the state is located at the center of the quadrotor at  $\xi$  and the diffeomorphic state is offset from  $\xi$  by a distance  $d$ .

The angular velocity vector in the quadrotor frame is related to the Euler angle rates by the following

$$\omega = {}_Q \omega_L = W(\eta) \dot{\eta} = \quad (49)$$

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \quad (50)$$

$$= \begin{bmatrix} \dot{\phi} - s_\theta \dot{\psi} \\ c_\phi \dot{\theta} + s_\phi c_\theta \dot{\psi} \\ -s_\phi \dot{\theta} + c_\phi c_\theta \dot{\psi} \end{bmatrix} \quad (51)$$

$$= \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \Rightarrow \quad (52)$$

$$W(\eta) = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \quad (53)$$



### 3.3.2 Euler-Lagrange Derivation

There are two main approaches for solving for the dynamics of a system (1) Newton-Euler and (2) Euler-Lagrange. Newton-Euler solves for the dynamics by using the law of conservation of momentum and Euler-Lagrange uses the law of conservation of energy. To solve for the dynamics using Euler-Lagrange one must solve the Euler-Lagrange equation

$$\frac{d}{dt} \left[ \frac{\partial L}{\partial \dot{q}} \right] - \frac{\partial L}{\partial q} = Q, \quad (54)$$

where  $Q$  is the external actions (forces / torques) and  $L$  is the Lagrangian, defined as the kinetic energy minus the potential energy,

$$L(q, \dot{q}) = T_{trans} + T_{rot} - U, \quad (55)$$

$$U = mgz, \quad (56)$$

$$T_{trans} = \frac{1}{2} m \dot{\xi}^T \dot{\xi}, \quad (57)$$

$$T_{rot} = \frac{1}{2} \omega^T I \omega = \frac{1}{2} \dot{\eta}^T W(\eta)^T I W(\eta) \dot{\eta} = \frac{1}{2} \dot{\eta}^T J(\eta) \dot{\eta}. \quad (58)$$

Here the rotational kinetic energy is defined with the Euler rates using  $J$ , which acts as an inertia matrix. Since, the Lagrangian contains no cross terms between the translational and rotational kinetic energies it can be decoupled and solved in two parts: translation and rotation.

### 3.3.2.1 Translational Component

The external force applied to the the quadrotor is always pointing in the negative  $z$ -direction of the quadrotor frame because the motors only rotate in one direction. Thus,

$$L_{trans} = T_{trans} - U = \frac{1}{2}m\dot{\xi}^\top \dot{\xi} - mg\xi_z, \quad (59)$$

$$\frac{d}{dt} \left[ \frac{\partial L_{trans}}{\partial \dot{\xi}} \right] - \frac{\partial L_{trans}}{\partial \xi} = {}_L F = \quad (60)$$

$$\left\{ \begin{array}{l} \frac{d}{dt} \left[ \frac{\partial L_{trans}}{\partial \dot{\xi}} \right] = \frac{d}{dt} [m\dot{\xi}^\top] = m\ddot{\xi}, \\ \frac{\partial L_{trans}}{\partial \xi} = -mg[0, 0, 1]^\top = -mgE_z \end{array} \right\} \Rightarrow \quad (61)$$

$$m\ddot{\xi} + mgE_z = {}_L F \Rightarrow \quad (62)$$

$$\ddot{\xi} = -gE_z - \frac{F}{m}RE_z. \quad (63)$$

### 3.3.2.2 Rotational Component

The external torque applied to the quadrotor comes from the moment generated from the force of each motor that is offset from the center of mass by the length of the arm and from the moment generated from the rotation of the propellers. The external torque as a vector in quadrotor frame can be written as

$$\tau = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} \frac{l}{\sqrt{2}}(F_2 + F_3 - F_1 - F_4) \\ \frac{l}{\sqrt{2}}(F_1 + F_3 - F_2 - F_4) \\ T_1 + T_2 - T_3 - T_4 \end{bmatrix}, \quad (64)$$

$$\tau_\eta = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = W(\eta)^{-1}\tau. \quad (65)$$

Thus,

$$L_{rot} = T_{rot} = \frac{1}{2} \dot{\eta}^T J(\eta) \dot{\eta}, \quad (66)$$

$$\frac{d}{dt} \left[ \frac{\partial L_{rot}}{\partial \dot{\eta}} \right] - \frac{\partial L_{rot}}{\partial \eta} = \tau_\eta = \quad (67)$$

$$\left\{ \begin{array}{l} \frac{d}{dt} \left[ \frac{\partial L_{rot}}{\partial \dot{\eta}} \right] = \frac{d}{dt} [\dot{\eta}^T J(\eta)] = J(\eta) \ddot{\eta} + \dot{J}(\eta) \dot{\eta}, \\ \frac{\partial L_{rot}}{\partial \eta} = \frac{1}{2} \frac{\partial \dot{\eta}^T J(\eta)}{\partial \eta} \dot{\eta} \end{array} \right\} \Rightarrow \quad (68)$$

$$J(\eta) \ddot{\eta} + \left( \dot{J}(\eta) - \frac{1}{2} \frac{\partial \dot{\eta}^T J(\eta)}{\partial \eta} \dot{\eta} \right) \dot{\eta} = \tau_\eta \Rightarrow \quad (69)$$

$$J(\eta) \ddot{\eta} + C(\eta, \dot{\eta}) \dot{\eta} = \tau_\eta \Rightarrow \quad (70)$$

$$\ddot{\eta} = J(\eta)^{-1} (\tau_\eta - C(\eta, \dot{\eta}) \dot{\eta}). \quad (71)$$

The  $C(\eta, \dot{\eta})$  function is known as the Coriolis term and can be deceptively difficult to calculate. This difficulty is because it is not obvious how to take the derivative of  $J$  with respect to time and  $\eta$ . These derivatives are shown here<sup>4</sup>

$$\dot{J} = \frac{\partial J}{\partial W} \dot{W} = 2W^T I \dot{W}, \quad (72)$$

$$\dot{W} = \frac{\partial W}{\partial \eta} \dot{\eta} \quad (73)$$

$$= \frac{\partial W}{\partial \phi} \dot{\phi} + \frac{\partial W}{\partial \theta} \dot{\theta} + \frac{\partial W}{\partial \psi} \dot{\psi} \quad (74)$$

$$= \begin{bmatrix} 0 & 0 & -c_\theta \dot{\theta} \\ 0 & -s_\phi \dot{\phi} & c_\phi c_\theta \dot{\phi} - s_\phi s_\theta \dot{\theta} \\ 0 & -c_\phi \dot{\phi} & -s_\phi c_\theta \dot{\phi} - c_\phi s_\theta \dot{\theta} \end{bmatrix} \quad (75)$$

and

$$M = \dot{\eta}^T J(\eta), \quad (76)$$

$$\frac{\partial M}{\partial \eta} = \begin{bmatrix} \frac{\partial M_1}{\partial \phi} & \frac{\partial M_2}{\partial \phi} & \frac{\partial M_3}{\partial \phi} \\ \frac{\partial M_1}{\partial \theta} & \frac{\partial M_2}{\partial \theta} & \frac{\partial M_3}{\partial \theta} \\ \frac{\partial M_1}{\partial \psi} & \frac{\partial M_2}{\partial \psi} & \frac{\partial M_3}{\partial \psi} \end{bmatrix}. \quad (77)$$

---

<sup>4</sup>Note,  $\frac{\partial W}{\partial \eta}$  is a tensor.

### 3.3.2.3 Equations of Motion

Therefore, the full state equation of motion for a quadrotor is

$$\dot{\mathbf{x}} = f(\mathbf{x}, u) = \begin{bmatrix} \dot{\xi} \\ \dot{\eta} \\ -gE_z - \frac{F}{m}RE_z \\ J(\eta)^{-1}(\tau_\eta - C(\eta, \dot{\eta})\dot{\eta}) \end{bmatrix}. \quad (78)$$

### 3.3.3 Linearization

Often for control and estimation it is beneficial (e.g. Linear Quadratic Regulator and Extend Kalman Filter) to have a linearization of the nonlinear system around a given operating point. Linearization gives a linear system for the dynamics of a perturbation of the state from the state operating point given a perturbation of the input from the input operating point. The linearization for a quadrotor is derived as follows

$$\dot{\mathbf{x}} = f(\mathbf{x}, u) = \left\{ \begin{array}{l} \mathbf{x} = \bar{\mathbf{x}} + \delta\mathbf{x} \\ u = \bar{u} + \delta u \end{array} \right\} \Rightarrow \quad (79)$$

$$\dot{\mathbf{x}} = \dot{\bar{\mathbf{x}}} + \dot{\delta\mathbf{x}} = f(\bar{\mathbf{x}} + \delta\mathbf{x}, \bar{u} + \delta u) = \{\text{Taylor Expand}\} \quad (80)$$

$$= f(\bar{\mathbf{x}}, \bar{u}) + \frac{\partial f}{\partial \mathbf{x}}(\bar{\mathbf{x}}, \bar{u})\delta(\mathbf{x}) + \frac{\partial f}{\partial u}(\bar{\mathbf{x}}, \bar{u})\delta(u) + \text{H.O.T} \Rightarrow \quad (81)$$

$$\dot{\delta\mathbf{x}} \approx \frac{\partial f}{\partial \mathbf{x}}(\bar{\mathbf{x}}, \bar{u})\delta(\mathbf{x}) + \frac{\partial f}{\partial u}(\bar{\mathbf{x}}, \bar{u})\delta(u) \Rightarrow \quad (82)$$

$$\dot{\delta\mathbf{x}} \approx A(\bar{\mathbf{x}}, \bar{u})\delta(\mathbf{x}) + B(\bar{\mathbf{x}}, \bar{u})\delta(u), \quad (83)$$

$$A(\bar{\mathbf{x}}, \bar{u}) = \frac{\partial f}{\partial \mathbf{x}}(\bar{\mathbf{x}}, \bar{u}), \quad (84)$$

$$B(\bar{\mathbf{x}}, \bar{u}) = \frac{\partial f}{\partial u}(\bar{\mathbf{x}}, \bar{u}). \quad (85)$$

### 3.3.4 Diffeomorphic State Dynamics

The linearization of the full state dynamics of a quadrotor is not completely controllable. This is easy to see in the equation of motions. When the rotation matrix between the quadrotor frame and the local fixed frame is identity then the input force,  $F$ , only affects the

$z$ -direction of the position dynamics. Since, the equation of motion for position and rotation are decoupled, the linearized rotation dynamics will have no affect on the position dynamics. This lack of control means there is no control action that can move the quadrotor in the  $x$ - or  $y$ -directions. However, a new state can be chosen to represent the state of the quadrotor that is completely controllable in the linearized dynamics. If the state of the quadrotor is represented at some distance  $d$  in its negative  $z$ -direction instead of at its center of mass, then the linearization of this new state will be completely controllable, which implies that the nonlinear dynamics of this diffeomorphic state are locally controllable. The diffeomorphic state can be thought of as the position of a ball that is at the end of a  $d$  length rod that is colinear to the quadrotor's  $z$ -axis. It is easy to imagine that this ball can be moved in all directions by the quadrotor. The equations of motion for the diffeomorphic state are almost the same as the equations of motion of the quadrotor. The rotational dynamics are the same and the position dynamics are similar, with just an addition of additional term that captures the movement of the diffeomorphic state position due to the rotation of the quadrotor. The dynamics are

$$\ddot{\zeta} = -gE_z - \frac{F}{m}RE_z + RD\alpha, \quad (86)$$

where  $\alpha$  is the angular velocity vector of the quadrotor with respect to the local fixed frame expressed in the quadrotor frame

$$\alpha = \dot{\omega} = \frac{d}{dt}(W(\eta)\dot{\eta}) = \dot{W}\dot{\eta} + W\ddot{\eta} \quad (87)$$

$$= \dot{W}\dot{\eta} + WJ^{-1}(\tau_\eta - C\dot{\eta}) \quad (88)$$

and

$$D = \begin{bmatrix} 0 & -d & 0 \\ d & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \quad (89)$$

These dynamics can be simplified if we expand out each term of the equation of motion

$$\ddot{\zeta}_x = -\frac{1}{m}r_{13}F + dr_{12}\alpha_x - dr_{11}\alpha_y \quad (90)$$

$$\ddot{\zeta}_y = -\frac{1}{m}r_{23}F + dr_{22}\alpha_x - dr_{21}\alpha_y \quad (91)$$

$$\ddot{\zeta}_z = -\frac{1}{m}r_{33}F + dr_{32}\alpha_x - dr_{31}\alpha_y - g, \quad (92)$$

where

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad (93)$$

and noting that

$$R E_z = \begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix}, \quad (94)$$

$$R D = \begin{bmatrix} dr_{12} & -dr_{11} & 0 \\ dr_{22} & -dr_{21} & 0 \\ dr_{32} & -dr_{31} & 0 \end{bmatrix}. \quad (95)$$

Factoring the dynamics we get

$$\ddot{\zeta} = \begin{bmatrix} -\frac{1}{m}r_{13} & dr_{12} & -dr_{11} \\ -\frac{1}{m}r_{23} & dr_{22} & -dr_{21} \\ -\frac{1}{m}r_{33} & dr_{32} & -dr_{31} \end{bmatrix} \begin{bmatrix} F \\ \alpha_x \\ \alpha_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}, \quad (96)$$

which is an affine system. Also, note that the dynamics of the diffeomorphic state do not depend on the  $\alpha_z$ , which makes sense because rotating around the  $z$ -axis of the quadrotor has no affect of the position of the diffeomorphic state. Combining the diffeomorphic state

with the yaw angle to form a differentially flat state, the dynamics of this state are

$$\begin{bmatrix} \dot{\zeta} \\ \dot{\psi} \\ \ddot{\zeta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0^{3 \times 3} & 0^{3 \times 1} & I^{3 \times 3} & 0^{3 \times 1} \\ 0^{1 \times 3} & 0^{1 \times 1} & 0^{1 \times 3} & I^{1 \times 1} \\ 0^{3 \times 3} & 0^{3 \times 1} & 0^{3 \times 3} & 0^{3 \times 1} \\ 0^{1 \times 3} & 0^{1 \times 1} & 0^{1 \times 3} & 0^{1 \times 1} \end{bmatrix} \begin{bmatrix} \zeta \\ \psi \\ \dot{\zeta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0^{4 \times 1} & 0^{4 \times 1} & 0^{4 \times 1} & 0^{4 \times 1} \\ -\frac{1}{m}r_{13} & dr_{12} & -dr_{11} & 0 \\ -\frac{1}{m}r_{23} & dr_{22} & -dr_{21} & 0 \\ -\frac{1}{m}r_{33} & dr_{32} & -dr_{31} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F \\ \alpha_x \\ \alpha_y \\ \alpha_z \end{bmatrix} + \begin{bmatrix} 0^{4 \times 1} \\ 0 \\ 0 \\ -g \\ 0 \end{bmatrix} \quad (97)$$

and can be written concisely as an affine system as

$$\ddot{\chi} = \bar{A}\chi + \bar{B}\mu + G. \quad (98)$$

To go from angular acceleration to attitude torques

$$\tau_\eta = JW^{-1}(\alpha - \dot{W}\dot{\eta}) + C\dot{\eta}. \quad (99)$$

### 3.3.5 Quadrotor Control

This section is dedicated to outlining the fundamentals of two different controllers: Linear Quadratic Regulator (LQR) and Controlled Lyapunov Function (CLF). In addition, simulation results of the implementation of these controllers on the quadrotor are shown.

#### 3.3.5.1 Linear Quadratic (LQ) Problem

A general linear system with quadratic cost has the dynamics of

$$\dot{x} = Ax + Bu, \quad (100)$$

and a quadratic cost objective over a finite time horizon of the form,

$$J(u) = \int_{t_0}^{t_f} \frac{1}{2}x(t)^\top Qx(t) + \frac{1}{2}u(t)^\top Ru(t)dt + \frac{1}{2}x(t_f)^\top Sx(t_f), \quad (101)$$

$$Q = Q^\top \succeq 0, S = S^\top \succ 0, R = R^\top \succeq 0. \quad (102)$$

The LQ problem is to find the input that minimizes the cost,

$$u^* = \arg \min_u J(u). \quad (103)$$

To solve this problem need tools from optimal control theory are needed, specifically the Hamilton-Jacobi-Bellman (HJB) theory is needed,

$$-\frac{\partial J^*}{\partial t} = \min_u \left( L(x, u) + \frac{\partial J^*}{\partial x} f(x, u) \right), \quad (104)$$

$$J^*(x, t_f) = \Psi(x), \quad (105)$$

where

$$f(x, u) = Ax + Bu, \quad (106)$$

$$L(x, u) = x^\top Qx + u^\top Ru, \quad (107)$$

$$\Psi(x) = x^\top Sx. \quad (108)$$

Plugging everything into the HJB and turning the math crank, the following for the input is derived (derivation shown in the Appendix in Appendix B.1),

$$u(t) = -R^{-1}B^\top P(t)x(t). \quad (109)$$

This is a stabilizing controller with a time varying gain. In (109)  $P$  is known as the costate and is the solution to the differential Riccati equation,

$$\dot{P} = PBR^{-1}B^\top P - PA - A^\top P - Q, \quad (110)$$

$$P(t_f) = S. \quad (111)$$

The differential Riccati equation can be solved numerically via backwards integration; noticed  $P$  is specified at the final time. In (110),  $P$  is connected to the cost,  $J$ , in the HJB by what is known as the value function or cost-to-go function,

$$J^* = \frac{1}{2}x^\top Px, P = P^\top \succ 0. \quad (112)$$



In summary, given a linear system, pick a cost by choosing weights for the state, input, and final state via the  $Q$ ,  $R$ , and  $S$  matrices. Solve the differential Riccati equation by integrating backwards from the final time to the initial time to get  $P$  (as a trajectory). Then use the costate,  $P$ , and the state,  $x$ , at time  $t$  to get the current input,  $u$ . Apply this input to apply to the system.

### 3.3.5.2 Linear Quadratic Regulator (LQR)

The Linear Quadratic Regular is a stabilizing constant gain feedback controller for a linear system with quadratic cost. The solution for LQR is one component of a larger class of problems from optimal control known as LQ (Linear Quadratic) problems, described in above in Section 3.3.5.1. The LQR solution gives the optimal constant feedback gains to stabilize the system given desired weighting matrices on the state and input.

It turns out that if the linear system is completely controllable (i.e. the controllability matrix,  $[A, AB, A^2B, \dots, A^{(n-1)}B]$ , is full rank) and the time horizon is taken out to infinity, then the result of turning the crank on the HJB in Section 3.3.5.1 is the algebraic Riccati equation instead of the differential Riccati equation,

$$0 = PBR^{-1}B^TP - PA - A^TP - Q. \quad (113)$$

Note here  $S$  from (104) is meaningless in the cost because time goes out to infinity, so there is no terminal time. Thus,  $S$  is dropped from the cost. The solution to the algebraic Riccati equation (ARE) is a constant  $P$  and can be solved analytically. This means the feedback gain is constant for all time,

$$u(t) = -R^{-1}B^TPx(t), \quad (114)$$

which is often expressed as

$$u(t) = -Kx(t), \quad (115)$$

$$K = R^{-1}B^TP, \quad (116)$$

where  $K$  is the constant feedback gain.

### 3.3.5.3 Quadrotor Attitude Control via Linearization and LQR

Given that the quadrotor dynamics are nonlinear, one might wonder how LQR is useful for a quadrotor because it is meant for linear systems, not a nonlinear system. A common technique for nonlinear systems is to linearize the system at each time point. Then use this linear system to solve the algebraic Riccati equation at each time point to get a new feedback constant gain; and, therefore, an input to apply to the system for the given time. An explanation of linearization is shown in Section 3.3.3. Fig. 8a and Fig. 8b show attitude stabilization for a quadrotor using an LQR controller.

### 3.3.5.4 Control-Lyapunov Function (CLF)

Alternatively to trying to make a nonlinear system linear and using linear control tools to control, one can use tools from nonlinear control theory directly for the control. The biggest hammer in the nonlinear control theorist's tool belt is Lyapunov theory. For an autonomous (no input) nonlinear system,  $\dot{x} = f(x)$ , Lyapunov theory says if you can find a *Lyapunov function*, a function that is always positive for all nonzero states, zero when the state is zero, continuously differentiable, and has a time derivative that is never positive, i.e.

$$V(x) > 0, \forall x \neq 0, \quad (117)$$

$$V(0) = 0, \quad (118)$$

$$V \in \mathcal{C}, \quad (119)$$

$$\dot{V} \leq 0, \forall x \in D \setminus 0, \quad (120)$$

$$\dot{V}(0) = 0, \quad (121)$$

then the system is stable (i.e. will not blow up) in the region  $D$ . If the time derivative is strictly negative then the system is asymptotically stable (i.e. will converge to zero) in  $D$ . There is a restriction on  $f$  that it must be locally Lipschitz (i.e. have finite derivative in  $D$ ). Thus, if given a system and a valid Lyapunov function then that system is a stable system for given region [24]

But how can Lyapunov theory be used for feedback control of nonautonomous system of the form  $\dot{x} = f(x, u)$ ?

The key to using Lyapunov theory for feedback control is to pick an input,  $u$ , that makes the time derivative of the candidate Lyapunov function strictly negative. A feedback controller that does this is called a Control Lyapunov Function (CLF).

### 3.3.5.5 Quadrotor Attitude Control via CLF

To create an attitude controller for the quadrotor using a CLF, one needs to first write out the nonlinear dynamics (derived in Section 3.3.2),

$$\ddot{\eta} = J(\eta)^{-1}(\tau_{\eta} - C(\eta, \dot{\eta})\dot{\eta}). \quad (122)$$

The dynamics must be rewritten to get in the form of  $\dot{x} = f(x, u)$  (i.e. as a single integrator),

$$u = \tau_{\eta}, \quad (123)$$

$$x_1 = \eta, \quad (124)$$

$$x_2 = \dot{\eta}, \quad (125)$$

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ J^{-1}(u - Cx_2) \end{bmatrix}. \quad (126)$$

Then pick a candidate Lyapunov function that satisfies the requirements listed above (actually this is not positive definite, so not really a Lyapunov function, but one can use LaSalle to show asymptotic stability),<sup>5</sup>

$$V(x) = \frac{1}{2}(x_1 + x_2)^T(x_1 + x_2). \quad (127)$$

---

<sup>5</sup>Finding a Lyapunov function that works is the hard part in this entire process. Many people devote their entire careers to figuring out how to pick the right Lyapunov function.

The next step is to differentiate  $V$  with respect to time, then pick the special input,  $u$ , value to make sure it is always negative,

$$\dot{V}(x) = (x_1 + x_2)^\top (\dot{x}_1 + \dot{x}_2) \quad (128)$$

$$= x_1^\top x_2 + x_2^\top x_2 + x_1^\top J^{-1}(u - Cx_2) + x_2^\top J^{-1}(u - Cx_2) \quad (129)$$

$$= \{u = J\hat{u} + Cx_2\} \quad (130)$$

$$= x_1^\top x_2 + x_2^\top x_2 + x_1^\top \hat{u} + x_2^\top \hat{u} \quad (131)$$

$$= \{\hat{u} = -(x_1 + 2x_2)\} \quad (132)$$

$$= -x_1^\top x_1 - 2x_1^\top x_2 - x_2^\top x_2 \quad (133)$$

$$= -(x_1 + x_2)^\top (x_1 + x_2) < 0. \quad (134)$$

The derivative of  $V$  is not negative definite but only negative semidefinite. However, using LaSalle Invariance Principle the system the the CLF feedback controller can still be shown to be stable [24]. LaSalle Invariance Principle says that if  $V$  is continuous and  $\dot{V}$  is negative semidefinite for all  $x \in \Omega$  then  $x$  will converge to the largest invariant set in  $V_0 = \{x \in \Omega | \dot{V}(x) = 0\}$ , meaning the largest set where both  $\dot{x}$  and  $\dot{V}(x)$  equal zero. Solving for the largest invariant set  $M$  can be done with the following,

$$V_0 = \{x \in \mathbb{R}^2 | \dot{V}(x) = 0\} \quad (135)$$

$$= \{x \in \mathbb{R}^2 | x_1 = x_2\} \quad (136)$$

$$\dot{x} = 0 \Rightarrow x_2 = 0 \Rightarrow \quad (137)$$

$$M = \{x \in \mathbb{R}^2 | x_1 = 0, x_2 = 0\}. \quad (138)$$

Thus, the origin is asymptotically stable with the following stabilizing CLF feedback controller

$$u = J\hat{u} + Cx_2 \quad (139)$$

$$= J(-(x_1 + 2x_2) + Cx_2) \quad (140)$$

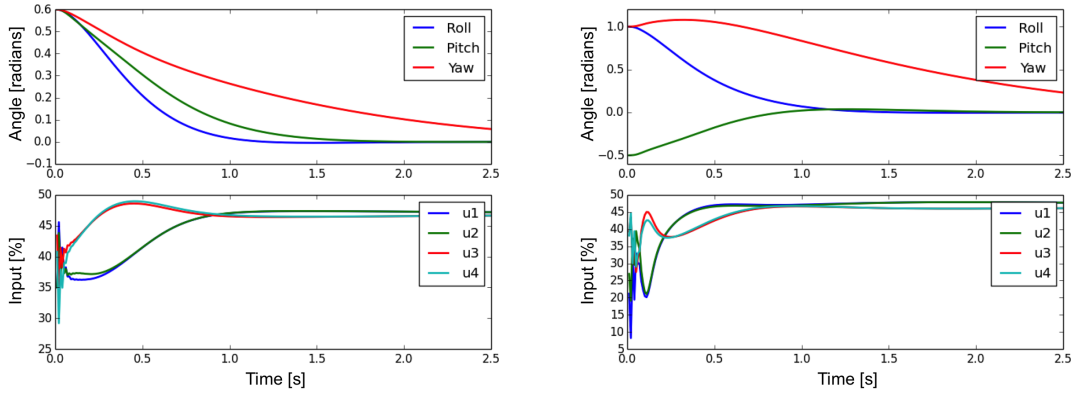
$$= -Jx_1 - 2Jx_2 + Cx_2 \quad (141)$$

$$= -Jx_1 + (C - 2J)x_2 \quad (142)$$

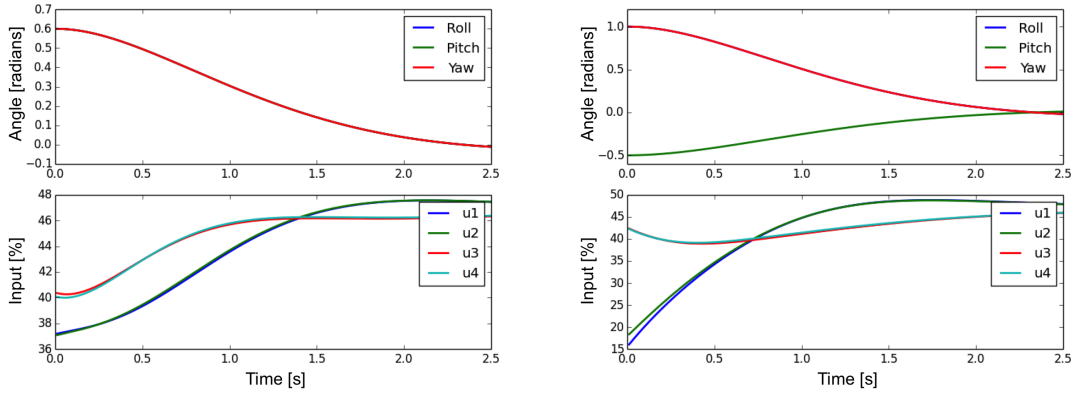
$$= \begin{bmatrix} -J \\ C - 2J \end{bmatrix} x. \quad (143)$$

It turns out this specific CLF is equivalent to a feedback linearization controller. This can be seen because the input is plugged back into the system it becomes a linear autonomous system.

The plots in Fig. 8 show the results for stabilizing the attitude of a quad from two different initial conditions using both an LQR and a CLF feedback controller. One should take notice how smooth the input trajectory is in the CLF controller verses to the LQR controller.



(a) LQR with initial conditions number 1. (b) LQR with initial conditions number 2.



(c) CLF with initial conditions number 1. (d) CLF with initial conditions number 2.

Figure 8: These plots show the response from an LQR and CLF feedback controller to stabilize the attitude of a quad from two different initial conditions. Initial condition 1: roll = 0.6, pitch = 0.6, and yaw = 0.6. Initial condition 2: roll = 1.0, pitch = -0.5, and yaw = 1.0.

### 3.4 Experimental Results

Experiments were conducted with the  $E^3$  algorithm for both the differential drive robot and the quadrotor. Limitations on the vertical space in which the motion capture system could see the quadrotor constrained the real quadrotor to only explore in 2 dimensions. The full 3 dimensional exploration was performed in simulation. To limit the real quadrotor to only in explore in 2 dimensions the information distribution was only applied to the first two states

of the system, similar to what was done with the differential drive robot to eliminate the heading angle state.

### 3.4.1 Differential Drive Robot Experimental Results

The  $E^3$  algorithm was run 100 times in simulation with a variety of different underlying unknown information distributions and was compared against two other algorithms. The first algorithm is the same as the  $E^3$  algorithm without the ergodic optimization portion; it simply uses the initial trajectory,  $u_i^0(t)$  (see Section 2.3.1) of the algorithm. The second algorithm uses random trajectories to explore the space. It was infeasible to test the EEDI algorithm for this problem since calculating the expected Fisher information matrix with  $C = 36$  would require a matrix of  $36^4$  (approximately 1.68 million) elements, with each element requiring an associated density function.

The algorithms were compared using a metric that sums information gain and effort in the following way

$$M = \int_0^{t_f} \int_X Q_M H(x) dx + \frac{1}{2} u(t)^\top R_M u(t) dt, \quad (144)$$

where  $Q_M \in \mathbb{R}$  penalizes the information uncertainty and  $R_M \in \mathbb{R}^{m \times m}$  penalizes the robot's input effort. In the results,  $Q_M = 3$  and  $R_M = \text{diag}([1, 1])$ , demonstrating that information gain is three times more important than effort. Table 3 show the average and standard deviation of the cost for each of the algorithms over the 100 trials.

The  $E^3$  algorithm had a significant improvement over random exploration, which was not surprising. In addition, using the locally optimal ergodic optimization demonstrated an improvement over the initial trajectory generated as an heuristic for the exploration trajectory. The results for running the  $E^3$  algorithm on the differential drive robot are shown in Fig. 9 and Fig. 10, where the latter figures illustrates each iteration of the  $E^3$  algorithm.

In robotics there is always a gap between simulation and reality. This gap can be seen in the results shown in Fig. 10, where the true trajectory of the robot does not exactly match

Table 3: Simulation Results: Mean Cost For 100 Trials

$E^3$ Algorithm	Initial Traj. Only	Random Traj.
$705 \pm 42$	$780 \pm 45$	$1071 \pm 87$

the locally optimal ergodic trajectory. This error is most notably due to acceleration not being accounted for in the dynamics of the robot and due to the motion capture system losing track of the robot, which causes erroneous motor commands. However, even with the true trajectory not exactly matching the locally optimal ergodic trajectory, the robot still effectively explores the space and was ergodic with respect to the underlying unknown information distribution, as seen in Fig. 9b.

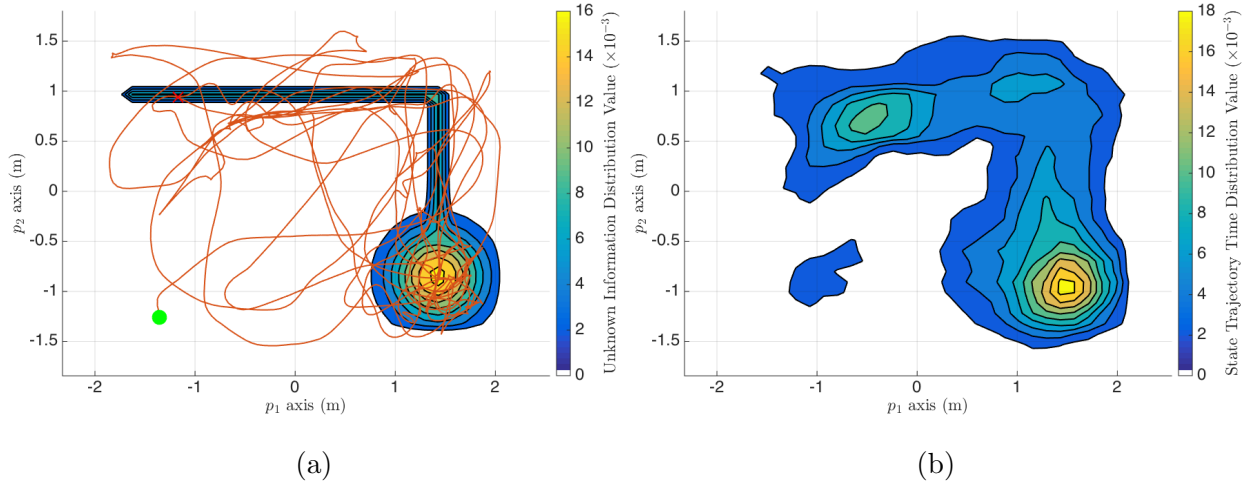


Figure 9: Experimental results of running the  $E^3$  algorithm on a differential drive robot exploring an area with an underlying unknown information distribution. (a) Underlying unknown information distribution and exploration trajectory of the robot (solid line) from the start location (dot) to the final location (X). (b) Distribution of differential drive robot locations over time while exploring the space.



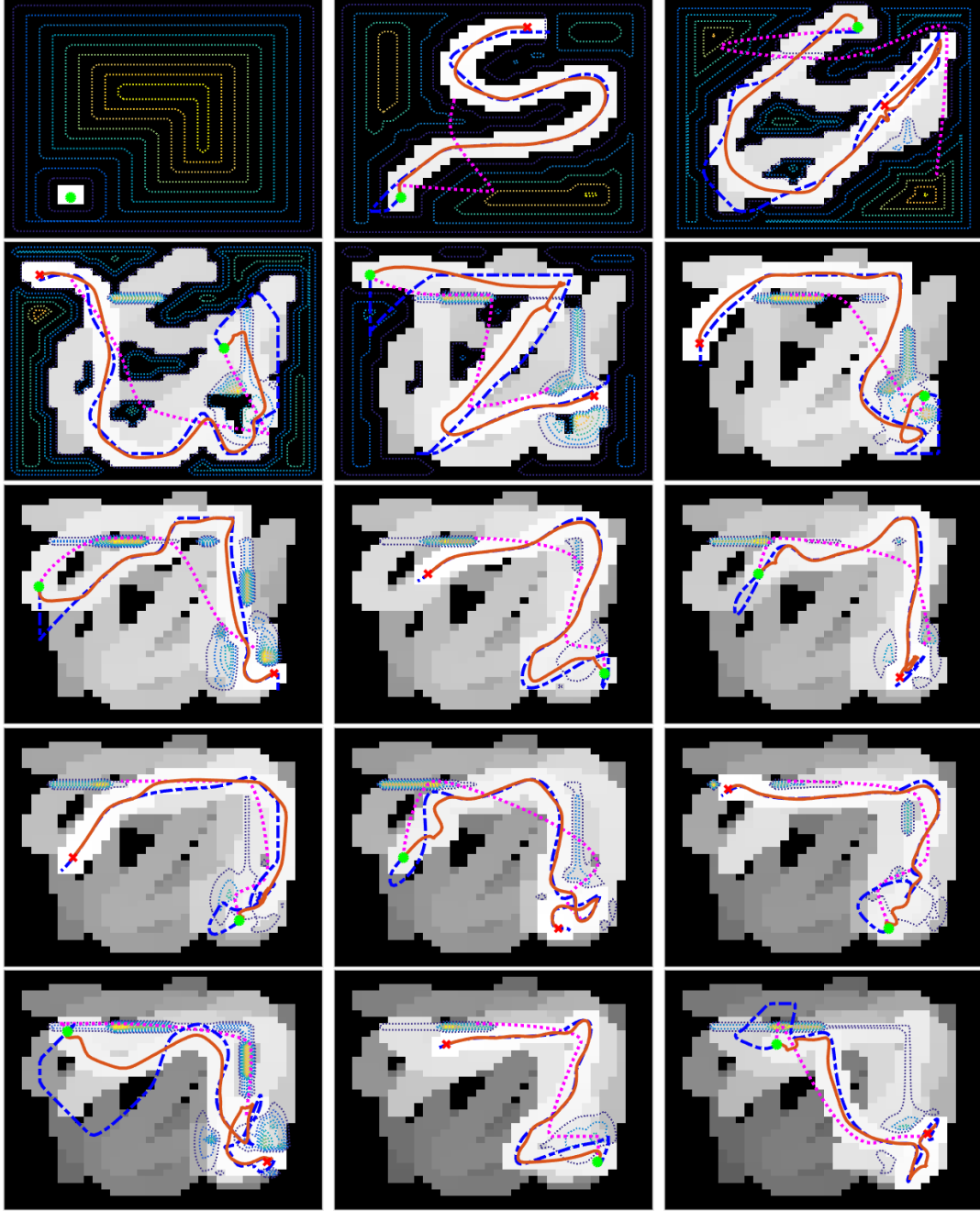
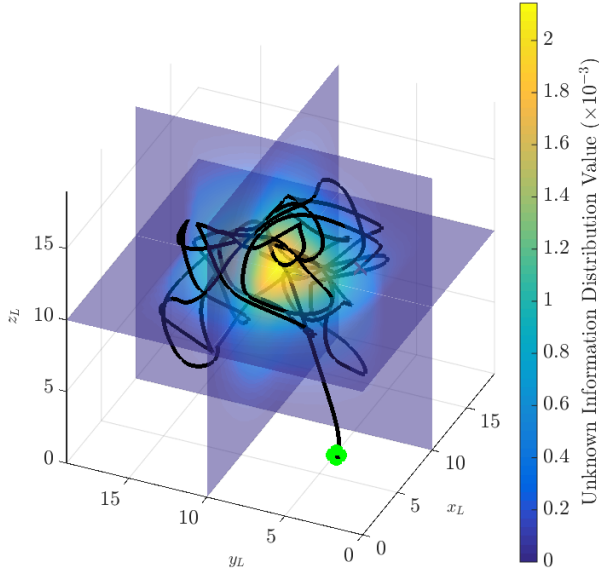


Figure 10: Evolution of  $E^3$  algorithm implemented on the Khepera differential drive robot exploring a 2-dimensional space with an underlying unknown information distribution. Each sub-figure shows the confidence value of the occupancy grid (black-gray-white regions), information gain map (dotted contour lines), initial trajectory (dotted line), locally optimal trajectory (dashed line), and actual trajectory the robot followed (solid line) from its start location (dot) to stop location (X).

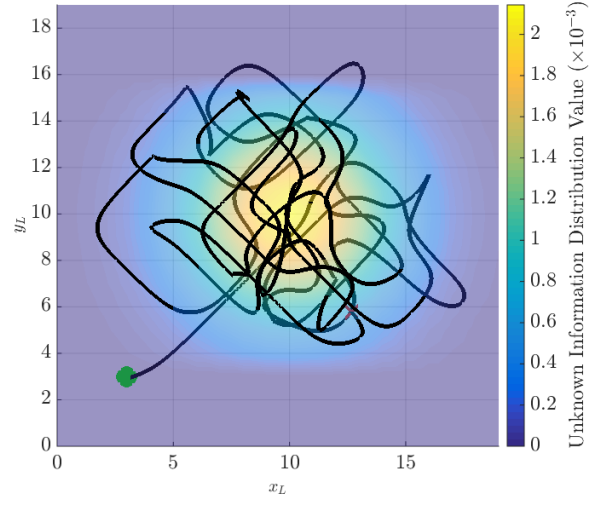
### 3.4.2 Quadrotor Experimental Results

Similar to the differential drive robot, the  $E^3$  algorithm was implemented on a quadrotor with a variety of different distributions. The real quadrotor was unable to perform 3-dimensional exploration in the lab due to limitation in vertical movement allowed by the motion capture system. Thus, 3-dimensional exploration was only performed in simulation and 2-dimensional exploration was performed on the real quadrotor.

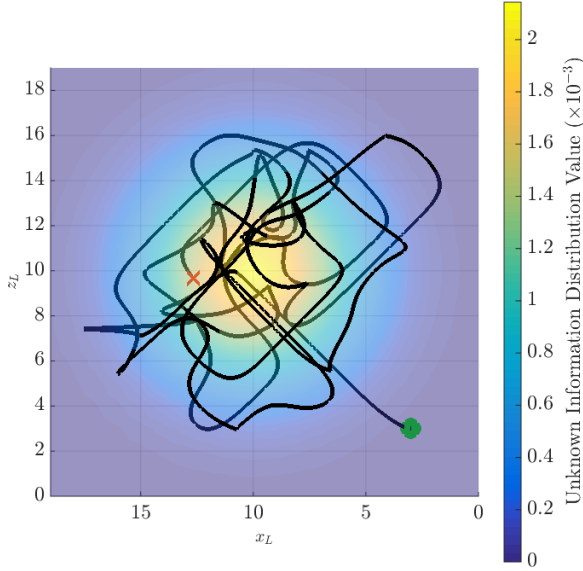
In the simulation experiments, the quadrotor was tasked with exploring a space of 20 cubic meters. The unknown information density is normally distributed with its mean at the center of the volume and diagonal covariance matrix with values of 10 for the entries. The quadrotor initially begins at the location  $[3, 3, 3]^T$  and explores the space for 400 seconds. Results of the final trajectory and unknown information distribution are shown in Fig. 11.



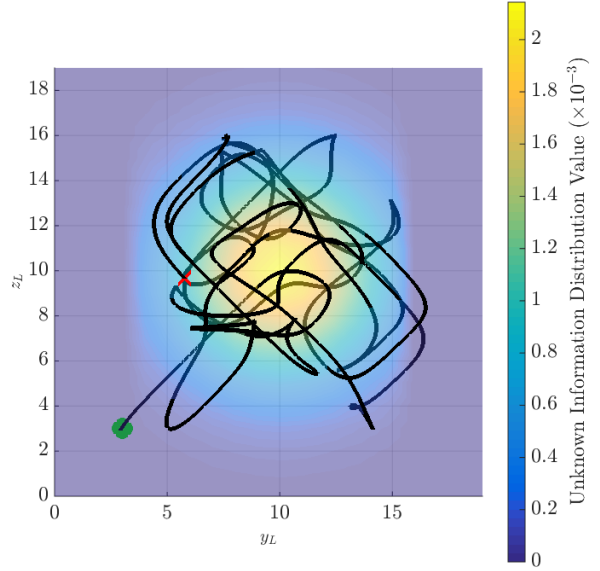
(a) 3D perspective



(b) XY-plane



(c) XZ-plane



(d) YZ-plane

Figure 11: Plots of simulation results using the E3 algorithm to have a quadrotor explore a 3D space. All four plots are of the same data from different perspectives. The black line is the quadrotor's trajectory, starting from the green circle and ending at the red cross.

The results for the real quadrotor are presented in the same form as presented for the differential drive robot in Section 3.4.1. Fig. 12 shows the complete trajectory of the quadrotor and the distribution of the trajectory from a top-down perspective. Fig. 13 shows the evolution of the algorithm, also from a top-down perspective.

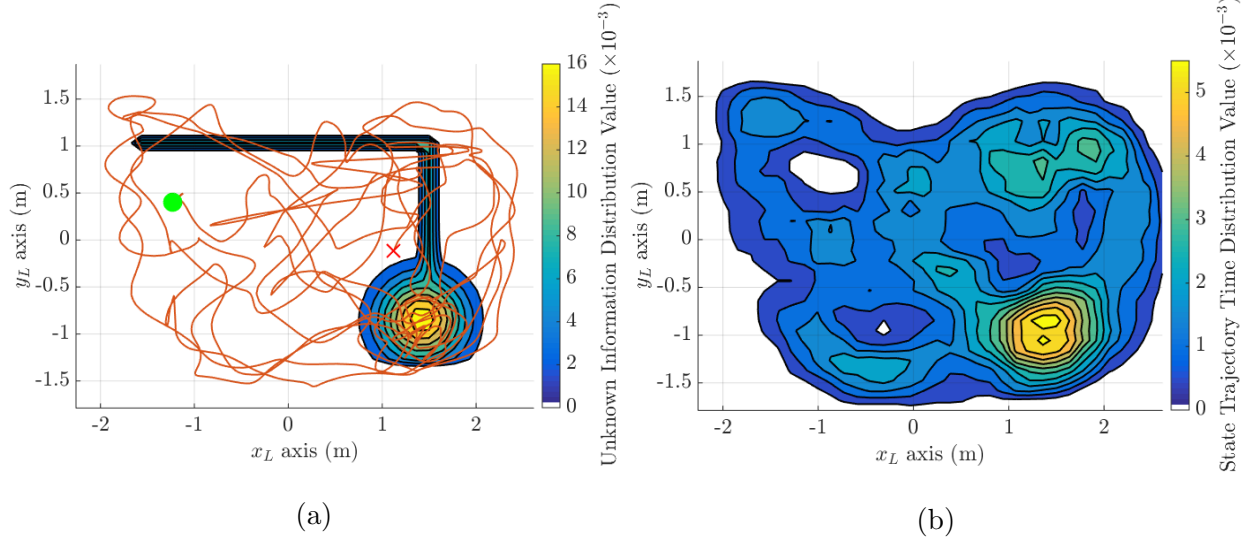


Figure 12: Experimental results of running the  $E^3$  algorithm on a quadrotor exploring an area with an underlying unknown information distribution. (a) Underlying unknown information distribution and exploration trajectory of the quadrotor (solid line) from the start location (dot) to the final location (X). (b) Distribution of quadrotor locations over time while exploring the space.

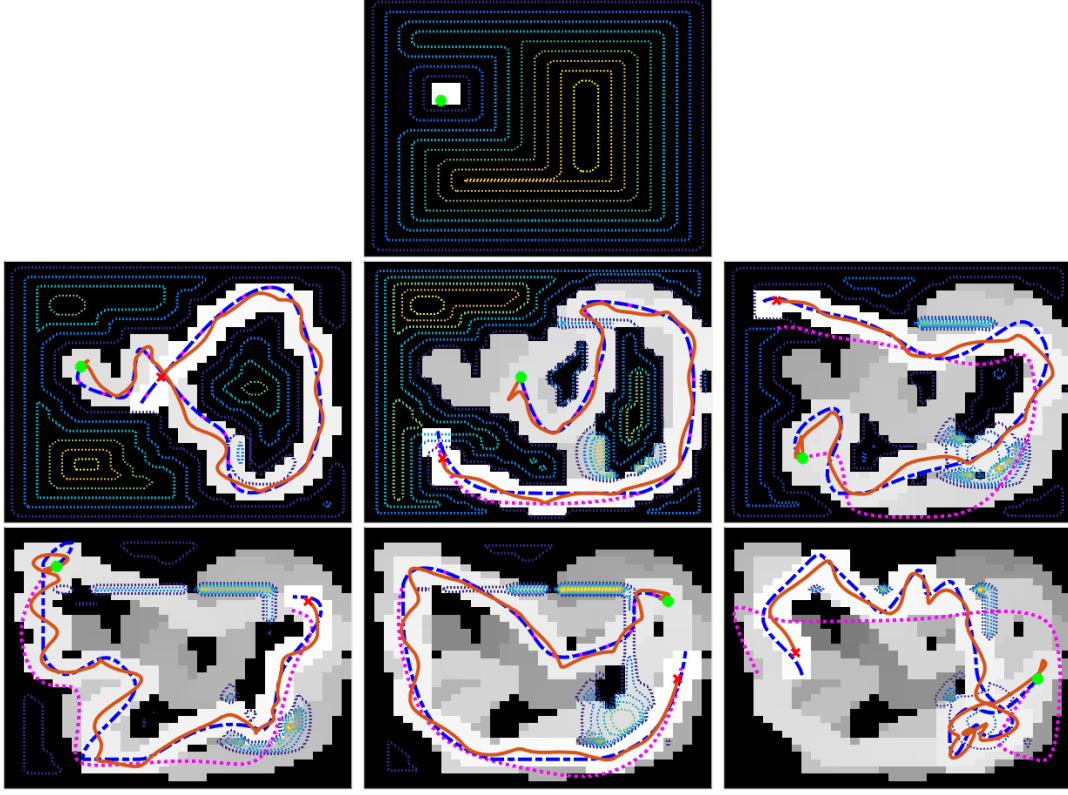


Figure 13: Evolution of  $E^3$  algorithm implemented on the quadrotor exploring a space with an underlying unknown information distribution. Each sub-figure shows the confidence value of the occupancy grid (black-gray-white regions), information gain map (dotted contour lines), initial trajectory (dotted line), locally optimal trajectory (dashed line), and actual trajectory the robot followed (solid line) from its start location (dot) to stop location (X).

### 3.5 Conclusion

These last two chapters described the theory and experimental results of a an agent learning an underlying information distribution in order to optimally explore an unknown space. The results of two types of experiments for optimal exploration were shown; the first involved a ground robot and the second involved an aerial robot. What if the same ground robot was tasked with exploring not only a 2 dimensional plane, over which it can navigate and measure the distribution, but also tasked with exploring a 3 dimension volume? If the ground robot was equipped with a sensor to measure the information density in 3 dimensions, then this

would be possible. Otherwise, it is easy to see that the ground robot does not have the necessary actuators (because it is a ground robot) or the necessary sensory components to accomplish the given learning objective of exploring the 3 dimension volume.

To determine the type of robot that can accomplish a given optimal exploration task, we must consider a larger question: when can a given dynamical system succeed at a given learning objective? In other words, what is the “learnability” of a dynamical system? The next chapter defines “learnability” for a dynamical system with a given objective function. Given this definition, a theorem and proof are provided on when a linear system with a quadratic cost is “learnable”. Following this theorem, it becomes clear that the ground robot used in the experiments for optimal exploration would not succeed if it had to explore a 3 dimensional volume.

## CHAPTER IV

### LEARNABILITY

#### *4.1 Introduction to Learnability*

As robotic systems become more interconnected, complex, and large-scale, machine learning tools are used with increasing frequency to design controllers. This integration is typically in conjunction with more standard control design methodologies. However, it is not clear when such methods are appropriate or effective, and this ambiguity has to do, in part, with the lack of a formal characterization of when the solution to a problem is “learnable” by a particular system. In this chapter, we attempt to remedy this problem by proposing a formal definition for learnability from a system-theoretic vantage-point, and then apply this definition to different types of mobile robots.

In order to produce a definition of learnability that is both relevant and applicable, one must first understand what “learning” itself implies. In the Merriam-Webster dictionary [36], *to learn* is “to gain knowledge or understanding of a skill by study, instruction, or experience.” But, some skills are clearly not learnable by a particular system. For example, a wheeled ground robot cannot learn to fly no matter the sophistication of its algorithms. Or, a robot with no sensors whatsoever can not learn how to approach a landmark. These two rather extreme examples both hint at the roles that classic system theoretic notions such as controllability and observability might take on when defining learnability.

In the traditional machine learning community, learning has been analyzed and characterized in different settings, and a working definition [40,68], states that “A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ”. From this rather informal definition one can deduce that if the performance measure

never improves no matter what the experience might be, then the computer program can not learn. We will follow this intuitive notion when defining the learnability concept for dynamical systems.

The main contribution of this chapter is a definition of what it means for a dynamical system to have the learning capability to improve its performance relative to a given performance cost. We call this notion *learnability*. Armed with such a definition, one can test whether a given dynamical system can learn a given task; similar to how one can test whether a given system is controllable or observable. The condition for learnability differs from these other standard conditions in that not only does it depend on the dynamical model of the system, but also on the cost function that is associated with the task. This means that a given system might not have learnability for one type of task but does for another type.

The term “learnability” has been used in other contexts before. The notion of learnability is most widely used in the field of statistical learning theory, which is the mathematical analysis of machine learning. Campi and Vidyasagar define *probably approximately correct* (PAC) learnable for a function class if there exists an algorithm that PAC-learns that function. An algorithm is PAC if the probability of the generalized error of the hypothesis for a target function being greater than  $\epsilon$  goes to zero as time goes to infinity [9] [63]. This notion was proposed by Valiant in 1984 [60]. Distribution-free PAC learnability is equivalent to a finite VC-dimension, which is named after the foundation work done in learning theory by Vapnik and Chervonenkis [61]. Learnability is also used in other areas. For example, learnability is used in software testing and is defined in ISO 9126 [22] as attributes of a software product that have a tolerance on the effort required by its users to learn its application.

Within the domains of controls, robotics, and dynamical systems, related ideas have been pursued in the contexts of system identification and optimal control through the investigation of conditions under which system parameters can be identified or optimal solutions do exist [8]. For example, Vidyasagar and Karandikar use a learning theoretic approach for system identification and use PAC learning as a condition on which the system can be identified [62].



Optimal control is also related to reinforcement learning, which was developed in the artificial intelligence community. In fact, strong connections between optimal control and reinforcement learning are made by Sutton and Barto [54]. This is because both optimal control and reinforcement learning seek to find a policy that minimizes some cost function (or maximizes some reward function) for a given task. The difference is in the type of problems that each technique is better suited for solving. Given this connection it is reasonable to assume that a condition on when a system can learn depends on the cost function and the optimal control policy that minimizes that cost function. This is in fact what is used in this chapter to define learnability.

## ***4.2 Learnability Definition***

Even though the word “learnability” has been used in a number of different fields and has different (mostly informal) definitions, the semantics, which defines when knowledge can be gained, is consistent. The motivation behind this work is to produce a precise system theoretic definition based on the idea that a system is learnable if and only if the system is capable of both performing actions and observing outputs that relate to the cost function that the system is attempting to minimize. Particularly, the system will be able to acquire the necessary knowledge to complete its objective if and only if (i) there are different initial conditions that result in different optimal control input trajectories and (ii) there are different initial conditions that produce different perceived costs for at least some input signal. We use perceived cost because it is assumed that the system does not have direct access to its state information; therefore, it must estimate both its state and the cost that is associated with that estimated state.

The intuition for the first criterion is that if the best action to take from every initial condition gives the same terminal cost then there is nothing that the system is capable of doing to reduce the cost. Therefore, every initial condition will have the same optimal control input trajectory. Thus, learning can not take place because the performance is not

being improved with added experience. The intuition for the second criterion is that if the perceived cost is the same no matter where the system is located then no matter what the system does the cost will not be reduced. Again, learning will not be possible.

To make this more precise, consider a system whose dynamics are of the form

$$\dot{x} = f(x, u), \quad y = h(x, u), \quad z = g(x, u), \quad (145)$$

where  $x(t) \in \mathbb{X}$  is the state,  $u(t) \in \mathbb{U}$  is the input,  $y(t) \in \mathbb{Y}$  is the measurement, and  $z \in \mathbb{Z}$  is the signal-of-interest to the problem that is to be solved. The system dynamics are encoded by the function  $f$ , the sensors are encoded through  $h$ , and the signal-of-interest is encoded by the function  $g$ . The signal-of-interest is the signal that actually affects the cost. The space of input signals is denoted as  $\mathcal{U}$ . Moreover, given that the state can not be measured directly it must be estimated. Let us assume that the estimated state has been obtained through an observer, e.g.,

$$\dot{\hat{x}} = l(\hat{x}, y, u), \quad (146)$$

for some observer dynamics  $l$ . We will denote the estimated signal-of-interest as  $\hat{z} = g(\hat{x}, u)$ .

Learning only makes sense if something is actually supposed to be learnt, (i.e. there is a cost to be minimized). For instance, this cost could be total distance traveled by a robot towards a landmark, and so forth. In this chapter, the cost function that the system is minimizing is defined as

$$J(u) = \int_0^T \Lambda(u, z) dt + \Psi(z(T)), \quad (147)$$

where  $\Lambda : \mathbb{U} \times \mathbb{Z} \rightarrow \mathbb{R}_+$  is the instantaneous cost and  $\Psi : \mathbb{Z} \rightarrow \mathbb{R}_+$  is the terminal cost for the final time  $T$ . For our purposes it is convenient to include the initial state of the system  $x_0 = x(0)$  into the argument of the cost. Thus,  $J(u, x_0)$  denotes the cost defined over two arguments; the control input and the initial condition. Let us denote the minimizer to the cost for a given initial condition as

$$u_{x_0}^* = \arg \min_{u \in \mathcal{U}} J(u, x_0), \quad (148)$$

and we assume that this minimizer exists. Given that the state can not be measured directly and must be estimated, we denote the estimated cost (or perceived cost) as

$$\hat{J}(u, x_0) = \int_0^T \Lambda(u, \hat{z}) dt + \Psi(\hat{z}(T)). \quad (149)$$

With the above system definition we can formally define *learnability*.

**Definition 1** (Learnability). *The tuple  $(f, h, g, l, J)$  is said to be learnable if*

1. *the optimal input is influenced by the state, i.e.*

$$\exists x_0, x'_0 \text{ s.t. } u_{x_0}^* \neq u_{x'_0}^*, \quad (150)$$

2. *the cost is influenced by the output, i.e.*

$$\exists x_0, x'_0, u \text{ s.t. } \hat{J}(u, x_0) \neq \hat{J}(u, x'_0). \quad (151)$$

This definition may seem obvious, but it does capture the two essential ingredients, namely that the state impacts the optimal input (or actuation) and the output (or measurements) impacts the perceived cost.

### 4.3 A Simple Yet Illustrative Example

Let us demonstrate the learnability condition on a simple example. Imagine a caterpillar that is inching along on a thin straight branch that is lying on the ground. The caterpillar can only move along the length of the branch – let us call that direction the  $x$ -direction – and its head is turned to the side, so it can only see to the side of the branch – the  $y$ -direction. The caterpillar uses its eyes as perfect sensors to measure distances in the  $y$ -direction. Moreover, the caterpillar's goal is to reach a delicious green leaf high above in the tree that hangs directly over the caterpillar – the  $z$ -direction – at distance  $\eta$ . Will the caterpillar be able to learn the necessary actions to reach its desired meal?

Let us set up the problem as follows to answer our question; the dynamics are

$$\dot{x} = [1, 0, 0]^T u, \quad (152)$$

$$y = [0, 1, 0]x, \quad (153)$$

$$z = [0, 0, 1]x, \quad (154)$$

and the cost is

$$J(u, x_0) = \int_0^T \|u\|^2 dt + (z(T) - \eta)^2. \quad (155)$$

This cost says that the caterpillar would like to exert the least amount of energy while getting as close as possible (in terms of height off the ground) to the leaf. However, no matter where the caterpillar starts on that branch the optimal actions are to do nothing. This is because  $z = 0$  for all possible  $x$  values that the caterpillar can obtain. This implies that  $\Psi = \eta^2$  and therefore  $u^* = 0 \forall x_0$ . The first condition of learnability is violated. Therefore, learning is not possible. Let us look at the second condition (151) just for demonstration. For any given input that the caterpillar uses the cost will always be  $\hat{J}(u, x_0) = \hat{J}(u, x'_0)$ , this is because the measurements that the caterpillar is taking in the  $y$ -direction do not affect the cost. Therefore, the second condition is also violated. The caterpillar has no hope of learning how to get to that delicious green leaf.

#### ***4.4 Learnability for Linear Systems With Quadratic Cost***

The definition of learnability establishes a particularly direct result for when a linear system is learnable with respect to a quadratic cost. In this section we formulate this linear system problem for learnability and state the linear system with quadratic cost (LQ) learnability theorem.

#### 4.4.1 Problem Statement

Consider a continuous-time linear time-invariant system of the form

$$\begin{aligned} \dot{x} &= Ax + Bu, \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^k, \\ y &= Cx, \quad y \in \mathbb{R}^m, \quad m \leq n, \end{aligned} \tag{156}$$

where  $x$  is the state,  $u$  is the control signal, and  $y$  is the measurement. In addition, let us define the signal-of-interest as

$$z = Gx, \quad z \in \mathbb{R}^l, \quad l \leq n. \tag{157}$$

Furthermore, let us assume that an estimate of the state is accomplished using a Luenberger observer of the form

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x}). \tag{158}$$

Therefore, our estimated signal-of-interest is

$$\hat{z} = G\hat{x}. \tag{159}$$

We do not want to impose unnecessary constraints on the choice of observer gain in (158); however, we do need to assume that the observer is not harmful in the sense that the observer should reflect different state estimates (after a while) if the difference between two different initial states are observable. In other words, we assume that<sup>1</sup>

$$\mathcal{R}(C^\top) \subseteq \mathcal{R}(L), \tag{160}$$

which is sufficient to ensure that at least some initial conditions on the state result in different state estimates.

The objective is for the system to learn how to transition from  $z(t)$  (a point in  $\mathbb{R}^l$  corresponding to the current output) to another point  $\eta$  (a desired point in  $\mathbb{R}^l$ ) with minimal

---

<sup>1</sup> $\mathcal{R}()$  and  $\mathcal{N}()$  denote the range space and null space, respectively.

input. The cost and estimated cost for the problem can be written as

$$J(u, x_0) = \int_0^T \|u\|^2 dt + (z(T) - \eta)^\top S (z(T) - \eta), \quad (161)$$

$$\hat{J}(u, x_0) = \int_0^T \|u\|^2 dt + (\hat{z}(T) - \eta)^\top S (\hat{z}(T) - \eta), \quad (162)$$

where  $x_0 = x(0)$ ,  $T$  is the in final time, and  $S = S^\top \succ 0$  is a weighting on the distance to the goal for each output component as compared with the input signal.

#### 4.4.2 Linear System With Quadratic Cost Learnability Theorem

Here we state the theorem for learnability on a linear system given a quadratic cost. Let us first introduce some notation. The controllability Gramian is denoted as  $\Gamma$  and the observability Gramian is denoted as  $\Omega$  and are defined as

$$\Gamma = \int_0^T e^{As} B B^\top e^{A^\top s} ds, \quad (163)$$

$$\Omega = \int_0^T e^{A^\top s} C^\top C e^{As} ds. \quad (164)$$

**Theorem 1** (Learnability). *A linear system with quadratic cost is learnable if<sup>2</sup>*

$$\mathcal{R}(G^T) \subseteq \mathcal{R}(\Gamma) \cap \mathcal{N}(\Omega)^\perp \text{ and } G \neq 0. \quad (165)$$

Theorem 1 declares that the states that affect the output must be states that are both controllable and observable. However, it is worth noting that learnability does not imply complete controllability or complete observability. A partially controllable and partially observable linear system may still be learnable. And, one might argue that this theorem is almost a truism, i.e., that it states almost exactly what one would expect. This is indeed the case and we do not take the lack of any major surprises here as a negative. Rather, it shows that the definition of learnability coincides well with what learnability should indeed mean.

---

<sup>2</sup>Please refer to Theorem 4.4.3 for the proof of Theorem 1.

#### 4.4.3 Linear System With Quadratic Cost Learnability Proof

To prove Theorem 1 it must be shown that both conditions in Definition 1 will always be satisfied if (165) is true. The proof is divided into two parts: (i) the first condition of Definition 1 is satisfied by  $\mathcal{R}(G^T) \subseteq \mathcal{R}(\Gamma)$  and (ii) the second condition is satisfied by  $\mathcal{R}(G^T) \subseteq \mathcal{N}(\Omega)^\perp$ . For the first part of the proof we need an expression for the minimizer  $u_{x_0}^*$  and for the second part of the proof we need an expression for  $\hat{J}(u, x_0) - \hat{J}(u, x'_0)$ .

Before we begin let us write the controllability and observability Gramians, respectively, as

$$\Gamma = K(T)[K^*(t)], \quad (166)$$

$$\Omega = M(T)[M^*(t)]. \quad (167)$$

In (166)  $K(t)$  is a linear operator such that

$$K(t)[u(t)] = \int_0^t e^{As} Bu(s) ds, \quad (168)$$

and  $K^*(t)$  is the adjoint of  $K$ ,

$$K^*(t) = B^\top e^{A^\top t}. \quad (169)$$

Additionally, in (167)  $M(t)$  is a linear operator such that

$$M(t)[u(t)] = \int_0^t e^{A^\top s} C^\top u(s) ds \quad (170)$$

and  $M^*(t)$  is the adjoint of  $M$ ,

$$M^*(t) = Ce^{At}. \quad (171)$$

##### 4.4.3.1 First condition

Let us solve for the minimizer  $u_{x_0}^*$  by using a method of projections in a Hilbert space. Egerstedt and Martin outline this method in more detail in [13]. We will begin by defining

a Hilbert space as the combination of two inner product spaces  $\mathcal{H} : L_2 \times \mathbb{R}_S^l$ , where

$$L_2 : \langle v, w \rangle_{L_2} = \int_t^T v^\top w d\tau, \quad (172)$$

$$\mathbb{R}_S^l : \langle p, q \rangle_{\mathbb{R}_S^l} = p^\top S q, \quad (173)$$

i.e.

$$\langle (v; p), (w; q) \rangle_{\mathcal{H}} = \langle v, w \rangle_{L_2} + \langle p, q \rangle_{\mathbb{R}_S^l}. \quad (174)$$

Here we are using  $v$  and  $w$  as dummy variables for control signals in  $\mathcal{U}$  and  $p$  and  $q$  as dummy variables for points in  $\mathbb{R}^l$ . We know that

$$x(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}Bu(s)ds = \beta(x_0) + Ku, \quad (175)$$

where  $K$  is a linear operator on  $u$  as it is defined in (168) and  $\beta(x_0) = e^{At}x_0$ . We can then express the output as

$$z = G\beta + GK u, \quad (176)$$

where we will drop the dependence on  $t$  and  $x_0$  for now. Let us define an affine variety  $V_\beta \subseteq \mathcal{H}$ , where

$$V_\beta = \{(u; z) \mid GK u - z = -G\beta\}. \quad (177)$$

Now, we can reformulate the minimization of (148) for the linear system in (156) and the cost in (161) as

$$\begin{aligned} \min_{(u; z) \in V_\beta} \|u\|_{L_2}^2 + \|z - \eta\|_{\mathbb{R}_S^l}^2 \Rightarrow \\ \min_{(u; z) \in V_\beta} \|(u; z) - (0; \eta)\|_{\mathcal{H}}^2. \end{aligned} \quad (178)$$

Let us define  $p \equiv (0; \eta) \in \mathcal{H}$ , which is the point in the Hilbert space  $\mathcal{H}$  with the absolute minimum value to (178). However,  $p$  does not satisfy the dynamics of the system (i.e.  $p \notin V_\beta$ ). The projection of  $p$  onto  $V_\beta$  (a point that satisfies the dynamics of the system) will be the point in  $V_\beta$  that minimizes (178), which is also defines the optimal  $u^*$  for (148). This projection is done in four steps:



1. Find the subspace  $V_0$  that is parallel to  $V_\beta$ ,

$$V_0 = \{(u; z) \mid GK u - z = 0\}.$$

2. Find the subspace  $V_0^\perp$  that is perpendicular to  $V_0$ ,

$$V_0^\perp = \{(w; \rho) \mid \langle (w; \rho), (u; z) \rangle_{\mathcal{H}} = 0, (u; z) \in V_0\}.$$

Which is rewritten by using the fact that we can write the inner product equation as

$$\langle (w; \rho), (u; z) \rangle_{\mathcal{H}} = 0 \quad \Leftrightarrow$$

$$\langle w, u \rangle_{L^2} + \langle \rho, z \rangle_{\mathbb{R}_S^l} = 0 \quad \Leftrightarrow$$

$$\langle w, u \rangle_{L^2} + \langle \rho, GK u \rangle_{\mathbb{R}_S^l} = 0 \quad \Leftrightarrow$$

$$\langle w, u \rangle_{L^2} + \langle S\rho, GK u \rangle_{\mathbb{R}^l} = 0 \quad \Leftrightarrow$$

$$\langle w, u \rangle_{L^2} + \langle K^* G^\top S\rho, u \rangle_{L^2} = 0 \quad \Leftrightarrow$$

$$\langle w + K^* G^\top S\rho, u \rangle_{L^2} = 0 \quad \Leftrightarrow$$

$$w + K^* G^\top S\rho = 0.$$

Thus,

$$V_0^\perp = \{(w; \rho) \mid w = -K^* G^\top S\rho\}.$$

3. Find the affine variety  $V_p$  that is parallel to  $V_0^\perp$  and contains  $p$ . The affine variety  $V_p$  is  $V_0^\perp + p$ , or

$$\begin{aligned} V_p &= \{(w'; \rho') \mid w' = w, \rho' = \rho + \eta, (w; \rho) \in V_0^\perp\} \\ &= \{(w'; \rho') \mid w' = -K^* G^\top S(\rho' - \eta)\}. \end{aligned}$$

4. Find the intersection of  $V_\beta$  and  $V_p$ , which is the projection of  $p$  onto  $V_\beta$ ,

$$\begin{aligned}
V_\beta \cap V_p & \Leftrightarrow \\
GKu - z = -G\beta \mid_{u=-K^*G^\top S(z-\eta)} & \Leftrightarrow \\
GK(-K^*G^\top S(z-\eta)) - z = -G\beta & \Leftrightarrow \\
-G\Gamma G^\top S(z-\eta) - z = -G\beta & \Leftrightarrow \\
-G\Gamma G^\top Sz + G\Gamma G^\top S\eta - z = -G\beta & \Leftrightarrow \\
(G\Gamma G^\top S + I)z = G\Gamma G^\top S\eta + G\beta & \Leftrightarrow \\
z = (G\Gamma G^\top S + I)^{-1}(G\Gamma G^\top S\eta + G\beta). & 
\end{aligned}$$

Note that  $(G\Gamma G^\top S + I)$  is always positive definite.

Therefore, the minimizer  $u_{x_0}^*$  to (148) is

$$u_{x_0}^* = -K^*G^\top S((G\Gamma G^\top S + I)^{-1}(G\Gamma G^\top S\eta + G\beta(x_0)) - \eta), \quad (179)$$

remember we described  $\beta$  as a function of  $x_0$  in (175).

#### 4.4.3.2 Second condition

Now, let us derive the an expression for  $\hat{J}(u, x_0) - \hat{J}(u, x'_0)$ . Before we do, let us recall that  $x_0 \neq x'_0$  and  $\hat{z} = G\hat{x}$ . In addition, we will say that  $\hat{z}' = G\hat{x}'$ . So,

$$\begin{aligned}
\hat{J}(u, x_0) - \hat{J}(u, x'_0) & \Leftrightarrow \\
\int_0^T \|u\|^2 dt + (\hat{z}(T) - \eta)^\top S(\hat{z}(T) - \eta) - & \\
\int_0^T \|u\|^2 dt + (\hat{z}'(T) - \eta)^\top S(\hat{z}'(T) - \eta) & \Leftrightarrow \\
(\hat{z}(T) - \eta)^\top S(\hat{z}(T) - \eta) - (\hat{z}'(T) - \eta)^\top S(\hat{z}'(T) - \eta) & \Leftrightarrow \\
\hat{z}(T)^\top S\hat{z}(T) - \hat{z}'(T)^\top S\hat{z}'(T) - 2\eta^\top S(\hat{z}(T) - \hat{z}'(T)) & \Leftrightarrow \\
(\hat{z}(T) + \hat{z}'(T))^\top S(\hat{z}(T) - \hat{z}'(T)) - 2\eta^\top S(\hat{z}(T) - \hat{z}'(T)) & \Leftrightarrow \\
((\hat{z}(T) + \hat{z}'(T))^\top S - 2\eta^\top S)(\hat{z}(T) - \hat{z}'(T)) & \Leftrightarrow \\
(\hat{z}(T) + \hat{z}'(T) - 2\eta)^\top S(\hat{z}(T) - \hat{z}'(T)). & (180)
\end{aligned}$$

Notice that  $\hat{J}(u, x_0) - \hat{J}(u, x'_0)$  is only guaranteed to be zero for all  $\eta$  values if  $\hat{z}(T) - \hat{z}'(T) = 0$ .

Let us expand  $\hat{z}(T) - \hat{z}'(T)$  to show when this term equals zero. But first we will need an expression for  $\hat{x}(t)$ , which is estimated by the observer. We defined the express for the observer in (158), let us simplify that expression to

$$\dot{\hat{x}} = \hat{A}\hat{x} + Bu + LCx, \quad (181)$$

where  $\hat{A} = (A - LC)$ . Therefore,

$$\hat{x}(t) = e^{\hat{A}t}\hat{x}_0 + \int_0^T e^{\hat{A}(t-s)}(Bu(s) + LCx(s))ds.$$

From this we can get

$$\hat{z}(t) = G \left( e^{\hat{A}t}\hat{x}_0 + \int_0^T e^{\hat{A}(t-s)}(Bu(s) + LCx(s))ds \right). \quad (182)$$

Now we plug (182) into  $\hat{z}(T) - \hat{z}'(T)$  to get

$$\begin{aligned} \hat{z}(T) - \hat{z}'(T) &= \\ G \left( e^{\hat{A}T}\hat{x}_0 + \int_0^T e^{\hat{A}(T-s)}(Bu(s) + LCx(s))ds - \right. \\ &\quad \left. e^{\hat{A}T}\hat{x}_0 - \int_0^T e^{\hat{A}(T-s)}(Bu(s) + LCx'(s))ds \right) = \\ G \int_0^T e^{\hat{A}(T-s)}LC(x(s) - x'(s))ds. \end{aligned}$$

We can expand this further by using the expression for  $x(t)$  from (175),

$$\begin{aligned} \hat{z}(T) - \hat{z}'(T) &= \\ G \int_0^T e^{\hat{A}(T-s)}LC((\beta(x_0) + Ku) - (\beta(x'_0) + Ku))ds &= \\ G \int_0^T e^{\hat{A}(T-s)}LC(\beta(x_0) - \beta(x'_0))ds &= \\ G \int_0^T e^{\hat{A}(T-s)}LCe^{As}(x_0 - x'_0)ds &= \\ G \int_0^T e^{\hat{A}(T-s)}LM^*ds(x_0 - x'_0). \end{aligned} \quad (183)$$

We will condense (183) further to

$$\hat{z}(T) - \hat{z}'(T) = G\hat{M}M^*(x_0 - x'_0), \quad (184)$$

where  $\hat{M}$  is a linear operator such that

$$\hat{M}(t)[u(t)] = \int_0^t e^{\hat{A}(t-s)} Lu(s) ds. \quad (185)$$

Note that with our assumption on  $L$  from (160) we have that

$$\begin{aligned} \mathcal{R}(C^\top) &\subseteq \mathcal{R}(L) && \Leftrightarrow \\ \mathcal{R}(M) &\subseteq \mathcal{R}(\hat{M}) && \Leftrightarrow \\ \mathcal{R}(MM^*) &\subseteq \mathcal{R}(\hat{M}M^*). \end{aligned} \quad (186)$$

With the expression in (179), (180), (184), and (186) we have all the parts in place to prove Theorem 1, which we restate below.

**Theorem 1** (Learnability). *A system is learnable if*

$$\mathcal{R}(G^T) \subseteq \mathcal{R}(\Gamma) \cap \mathcal{N}(\Omega)^\perp \text{ and } G \neq 0. \quad (187)$$

*Proof.* Let us begin with  $\mathcal{R}(G^\top) \subseteq \mathcal{R}(\Gamma)$ , which implies the following

$$\begin{aligned} \mathcal{R}(G^\top) &\subseteq \mathcal{R}(\Gamma) && \Leftrightarrow \\ \mathcal{R}(G^\top) &\subseteq \mathcal{R}(K) && \Leftrightarrow \\ \mathcal{R}(G^\top) &\subseteq \mathcal{N}(K^*)^\perp && \Leftrightarrow \\ \exists \zeta \text{ s.t. } K^*G^\top S\zeta &\neq 0 && \Leftrightarrow \\ \exists x_0, x'_0 \text{ s.t. } u_{x_0}^* &\neq u_{x'_0}^*. \end{aligned}$$

Next,  $\mathcal{R}(G^\top) \subseteq \mathcal{N}(\Omega)^\perp$  implies the following

$$\begin{aligned}
\mathcal{R}(G^\top) &\subseteq \mathcal{N}(\Omega)^\perp && \Leftrightarrow \\
\mathcal{R}(G^\top) &\subseteq \mathcal{R}(\Omega^\top) && \Leftrightarrow \\
\mathcal{R}(G^\top) &\subseteq \mathcal{R}(MM^*) && \Rightarrow \\
\mathcal{R}(G^\top) &\subseteq \mathcal{R}(\hat{M}M^*) && \Rightarrow \\
\mathcal{R}(\hat{M}M^*) &\not\subseteq \mathcal{R}(G^\top)^\perp && \Leftrightarrow \\
\mathcal{R}(\hat{M}M^*) &\not\subseteq \mathcal{N}(G) && \Leftrightarrow \\
\exists \zeta \text{ s.t. } G\hat{M}M^*\zeta &\neq 0 && \Leftrightarrow \\
\exists x_0, x'_0 \text{ s.t. } G\hat{M}M^*(x_0 - x'_0) &\neq 0 && \Leftrightarrow \\
\exists x_0, x'_0 \text{ s.t. } \hat{z}(T) - \hat{z}'(T) &\neq 0 && \Leftrightarrow \\
\exists x_0, x'_0, u \text{ s.t. } \hat{J}(u, x_0) - \hat{J}(u, x'_0) &\neq 0 && \Leftrightarrow \\
\exists x_0, x'_0, u \text{ s.t. } \hat{J}(u, x_0) &\neq \hat{J}(u, x'_0). && 
\end{aligned}$$

Thus,

$$\begin{aligned}
\mathcal{R}(G^\top) &\subseteq \mathcal{R}(\Gamma) \cap \mathcal{N}(\Omega)^\perp \text{ and } G \neq 0 \Rightarrow \\
\exists x_0, x'_0, u \text{ s.t. } u_{x_0}^* &\neq u_{x'_0}^* \text{ and} \\
\exists x_0, x'_0, u \text{ s.t. } \hat{J}(u, x_0) &\neq \hat{J}(u, x'_0),
\end{aligned}$$

which satisfies the two conditions for learnability.  $\square$

## 4.5 Inverted Pendulum Robot

Let us give an example of learnability by determining if a two-wheel inverted pendulum robot can learn to move towards a goal. The dynamics of a two-wheel inverted pendulum robot can be found in [25] and [12] and an illustration of the system is shown in Fig. 14.

The state of this systems is

$$x = [x_1, x_2, v, \psi, \dot{\psi}, \phi, \dot{\phi}]^\top, \quad (188)$$

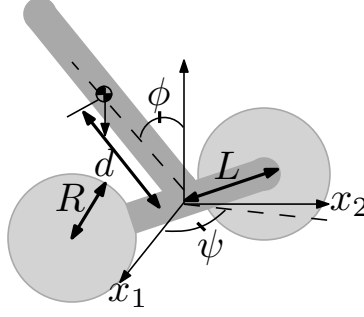


Figure 14: Illustration of two-wheel inverted pendulum robot [12].

where  $x_1$  and  $x_2$  is the position of the robot in the 2D plane,  $v$  is the forward velocity,  $\psi$  is the angle of the robot's orientation relative to the  $x_1$ -axis, and  $\phi$  is the angle of the robot's posture relative to the straight up unstable configuration.

The linearization of these dynamics around an operating point of

$$\bar{x} = [\bar{x}_1, \bar{x}_2, 0, \bar{\psi}, 0, 0, 0]^\top \text{ and } \bar{u} = [0, 0]^\top,$$

for a particular set of parameters that were used in [25] are shown here,

$$A = \begin{bmatrix} 0 & 0 & \cos(\bar{\psi}) & 0 & 0 & 0 & 0 \\ 0 & 0 & \sin(\bar{\psi}) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.16 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 72.49 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1.67 & -1.67 \\ 0 & 0 \\ 0.03 & -0.03 \\ 0 & 0 \\ -24.15 & -24.15 \end{bmatrix}.$$

We will say that the system has direct measurements of its  $x_1$  and  $x_2$  position, thus

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The cost function  $J(u, x_0)$  is the same as it is written in (161). It is straight forward to show that this is not completely controllable ( $\text{rank}(\Gamma) = 6$ ) and not completely observable ( $\text{rank}(\Omega) = 5$ ); however, that does not mean it is not learnable.

For this system the intersection of the controllable space and observable space is

$$\mathcal{R}(\Gamma) \cap \mathcal{N}(\Omega)^\top = \text{span} \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\}.$$

First, we will assume that the goal location is along the  $x_1$ -axis and that the robot is linearized around the orientation of  $\bar{\psi} = 0$ . In this case  $G$  is written as

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

and  $\mathcal{R}(G^\top)$  is obviously subset of  $\mathcal{R}(\Gamma) \cap \mathcal{N}(\Omega)^\top$ ; therefore, the system is learnable.

Now let us assume that the goal location is not on the  $x_1$ -axis and that the robot is still linearized around the orientation of  $\bar{\psi} = 0$ . In this case  $G$  is written as

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Here  $\mathcal{R}(G^\top)$  is not a subset of  $\mathcal{R}(\Gamma) \cap \mathcal{N}(\Omega)^\top$ ; therefore, the *linearized* system is not learnable.

This illustrative example demonstrates that if a system is not inherently linear then one must be careful in using The 1 to test for learnability with the linearization of the system. This is similar to how one must be careful in testing for controllability by using the linearization of the system.

## 4.6 Simple Example Revisited On A Real Robot

In Section 4.3 we show why a caterpillar is unable to learn how to get to a delicious green leaf when it is constrained to a branch on the ground. To further demonstrate this caterpillar

example we built a simple robot that captures the constrained controllable actions and observable measurements of the caterpillar. A picture of this robot is shown in Fig. 16.

This robot consists of a body and two moveable appendages. Each appendage has only one rotational degree of freedom that are parallel. This limits the robot to moving only forward and backward along a straight line. In addition, the robot has a sensor to measure distance. The objective of the robot is to learn how to move forward towards a goal that is some distance away.

The model we used to approximate the robot is a linear system of form used in Section 4.4.1. Let us use Theorem 1 to show when the robot can and can not learn. For this system

$$A = \mathbf{0}^{2 \times 2}, \quad B = [1, 0, ]^\top, \quad \text{and} \quad G = [1, 0].$$

Here the states are the  $(x, y)$  location of the robot in the ground plane. We have assumed the input drives the robot forward or backward along the  $x$ -axis and is not an input to each individual appendage. It is easy to see from these dynamics that

$$\mathcal{R}(G^\top) = \text{span}\{[1, 0]^\top\} \quad \text{and} \quad \mathcal{R}(\Gamma) = \text{span}\{[1, 0]^\top\}.$$

We ran two experiments where we changed the position of the sensor for the robot. In the first experiment we position the robot's sensor so that it is aligned with the direction of movement ( $x$ -axis). Thus,

$$C_1 = [1, 0] \quad \text{and} \quad \mathcal{N}(\Omega)^\perp = \text{span}\{[1, 0]^\top\}.$$

In the second experiment we position the robot's sensor so that it is aligned orthogonal to the direction of movement ( $y$ -axis). Thus,

$$C_2 = [0, 1] \quad \text{and} \quad \mathcal{N}(\Omega)^\perp = \text{span}\{[0, 1]^\top\}.$$

According to Theorem 1 the objective is learnable in the first experiment and not learnable in the second experiment. For both experiments we ran a standard reinforcement



learning algorithm, specifically Q-learning [54], and observed how well the robot learned to move towards the goal. The results are shown in Fig. 15.

In both experiments the robot is trying to learn how to reach a goal that is at 35 cm in the  $x$ -direction (robot's position is dark dash line). In the first experiment the robot explores for roughly the first 60 seconds of the experiment attempting to figure out which actions brings it closer to the goal. After exploring (dark areas of horizontal line) it starts exploiting (light areas of horizontal line) what it has learned and eventually reaches the goal. The total accumulated cost does not continue to increase (light gray dotted line). In the second experiment the robot continuously explores the different actions attempting to learn which action will consistently move it towards the goal. It is unable to learn the actions because it does not have the measurement information to inform it on which actions have moved it forward. The accumulated cost continues to increase <sup>3</sup>.

---

<sup>3</sup>A video of the physical robot learning can be viewed at <http://www.youtube.com/watch?v=cniTpI2oOjI>

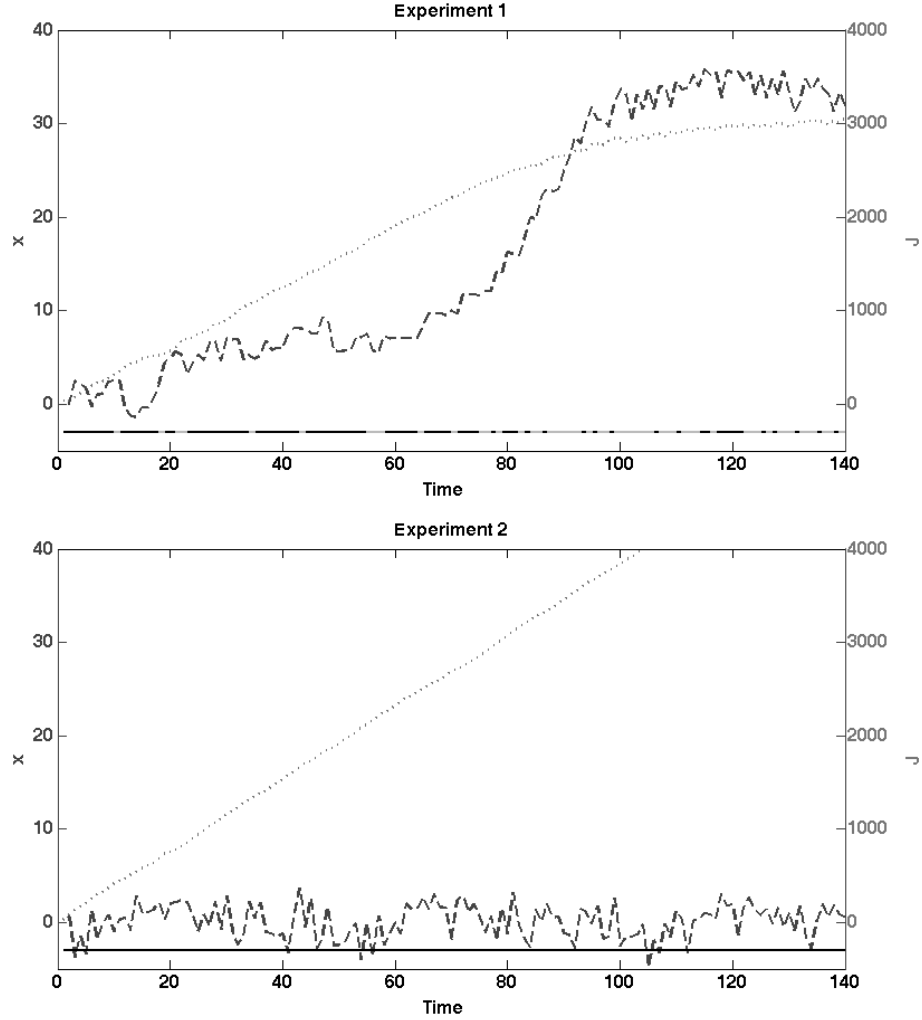


Figure 15: The top plot shows the results from experiment 1, which is when the robot is measuring distances in the direction it is moving. The bottom plot shows the results from experiment 2, which is when the robot is not measuring distances in the direction it is moving. The dashed black line is the distance the robot has moved. The dotted light gray line is the accumulated cost. The horizontal line along the bottom of the plots shows when the robot is exploring new learned actions (dark areas of the line) and exploiting previous learned actions (light areas of the line)

## 4.7 *Conclusion*

This chapter provides a formal definition for when a dynamical system can make progress to reduce the cost of a given objective function. This is equivalent to saying when can a dynamical system make progress in acquiring knowledge for a particular problem. This concept is defined as “learnability”. For a linear system with a quadratic cost, a theorem is provided that states whether the system is “learnable”. While the definition and associated learnability theorem identifies when a system can learn in theory, it does not mandate that the system can learn in practice. The difference in practice and theory is mostly attributed to the computational complexity of the system to explore the entire space; the space to explore is often too big to make any head way on the learning objective. In the next chapter, an algorithm is presented that reduces the size of large spaces so that learning can still take place.

## CHAPTER V

### LOW-DIMENSIONAL LEARNING FOR COMPLEX ROBOTS

#### *5.1 Introduction to Low-Dimensional Learning*

When control design is prohibitive due to complexities of the specifications and the systems themselves, machine learning provides a possible way forward. In fact, learning as a means to produce control strategies has been used on a number of complex systems, such as helicopters [3], humanoid robots [18, 28], robotic arms [52], biological systems [11], and wind turbines [30]. Despite the success associated with these particular applications, a hurdle that almost all learning algorithms face is the “curse of dimensionality”; coined by Richard Bellman in the 1950s. This is the exponential increase of information that must be learned as the number of possible states and actions in the system increases.

In this chapter, a model-free learning algorithm is presented that overcomes this complexity issue by a particular choice of discretization. The algorithm uses boundary conditions coupled with sets of primitive control laws to create motions for the locomotion of complex robotic systems. In particular, the presented algorithm learns actions based on boundary states instead of the actual system states, which greatly reduces the amount of learning that must take place.

To illustrate how this learning algorithm may be used, imagine a situation where a roboticist would like to build a robotic caterpillar, without having to (or even knowing how to) explicitly design the control actions to move the robot forward or backward. Instead, the roboticist simply “loads” the presented learning algorithm together with a library of primitive feedback controllers onto the robotic caterpillar. On its own, the robotic caterpillar learns to move forward and backward.

This chapter uses *reinforcement learning*, which is a subset of machine learning, where

an agent learns how to make a sequence of decisions to maximize some cumulative long-term expected reward; typically by interacting with the environment [54]. The scalability of reinforcement learning to high-dimensional continuous state-action systems can be problematic, as observed in [55]. The scalability problem derives from the fact that, in general, reinforcement learning is attempting to learn the best action to take for each state of the system (a state-action pair) based on a given reward function. In order to facilitate such a formulation, the state-space and action-space must be discretized, partitioned, or parameterized in some way. Unfortunately, the number of possible state-action pairs grows exponentially with the growth of both the state-space and the action-space dimensions. Reinforcement learning quickly becomes infeasible because its complexity scales linearly in the number of actions and quadratically with the number of states [27].

Previous work has been done to try to mitigate this problem. For example, Kuo et al. [31] discuss different sampling techniques that can be used for numerical integration in high dimensional spaces. In our work, we are not attempting to do numerical integration but face the same problem of feasibly performing some task in a high dimensional space. Most techniques in dealing with this problem of high dimensionality are tackled by sampling the space in an intelligent fashion. Zoppoli et al. do this with neural approximators [70]. Other techniques use clustering [19], function approximation [35], or adaptive dynamic programming with neural networks [66]. We cope with the problem in a similar fashion as is done in previous work; by sampling the space in an intelligent way. Our technique differs in that it hones in on the most important states of the state space and only addresses those states while ignoring the rest. These important states are the switching boundaries of the hybrid system.

The method outlined here to ameliorate this scalability problem is inspired by nature. It has been shown that animals and insects use a small set of motor primitives to construct and control movements [4, 26, 43, 56]. Moreover, the transitions between motor primitives do not occur everywhere in the state space and we interpret this switching between motor primitives

in terms of boundaries on which transitions may take place. This set of motor primitives suggest that the action-space can be reduced to a finite space where the dimension is equal to the cardinality of the primitive control set. In addition, the state-space used for reinforcement learning can also be reduced to a finite set of boundaries. Therefore, the reinforcement learning algorithm will only need to learn *boundary-controller pairs* instead of state-action pairs. The real strength with this approach is that for highly complex systems—particularly those where it is infeasible to formulate an accurate model of the system dynamics due to imprecise manufacturing, unknown material properties, or complex physical interactions (e.g friction and fluid dynamics)—control for locomotion may still be learned in a computationally feasible manner.

This work uses *reinforcement learning*, which is a subset of machine learning, where an agent learns how to make a sequence of decisions to maximize some cumulative long-term expected reward; typically by interacting with the environment [54]. The scalability of reinforcement learning to high-dimensional continuous state-action systems can be problematic, as observed in [55]. The scalability problem derives from the fact that, in general, reinforcement learning is attempting to learn the best action to take for each state of the system (a state-action pair) based on a given reward function. In order to facilitate such a formulation, the state-space and action-space must be discretized, partitioned, or parameterized in some way. Unfortunately, the number of possible state-action pairs grows exponentially with the growth of both the state-space and the action-space dimensions. Reinforcement learning quickly becomes infeasible because its complexity scales linearly in the number of actions and quadratically with the number of states [27].

The use of reinforcement learning with the reduced number of boundary-controller pairs presented in this chapter closely relates to Iterative Learning Control (ILC). ILC refines the input signal over repeated task iterations so that the output approaches the desired output for all points in the trajectory [41]. ILC in combination with adaptive switching of feedback gains has been used for control of robot manipulators with repetitive tasks [21, 44, 45, 53, 65]. Our

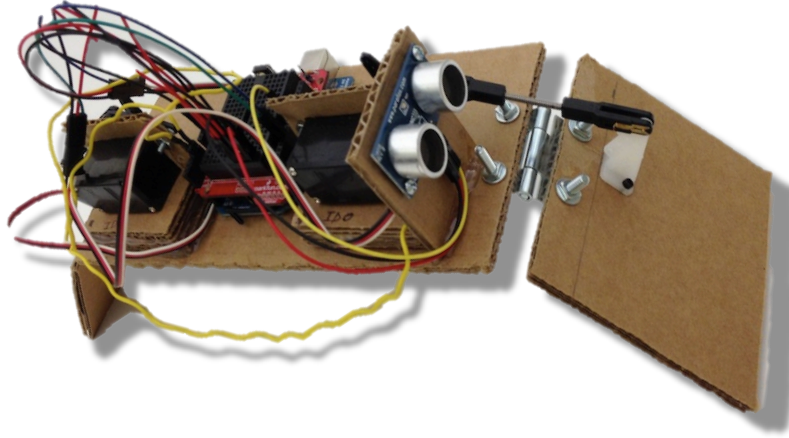


Figure 16: Physical robotic system used to test the learning algorithm.

algorithm differs in that it is not switching between different controller gains but completely different controllers. In addition, our algorithm may switch between controllers several times per cycle of the states instead of switching once per cycle or task execution.

The main contributions of this chapter are (i) the introduction of a hybrid system methodology and reinforcement learning algorithm to learn control actions based on boundary conditions to mitigate the “curse of dimensionality”, and (ii) the demonstration of the algorithm on both a simulated system and on a real robot shown in Fig. 16.

The chapter is organized in the following way. In Section 5.2 we describe the properties of a hybrid systems used for learning. In Section 5.3 we present the learning algorithm. In Section 5.4 we show results for two simulated systems and for a physical robotic system.

## 5.2 *System Overview*

This section introduces a learning algorithm for the locomotion of complex robotic systems. The form of the system that the learning algorithm is applicable to is outlined in detail below, but in general it is a continuous time system with the objective of moving in some direction. The algorithm described here learns the appropriate sequence of control laws and the switching protocol that produces a motion that moves the system the “best”, based on a cost function. The switching between the control actions occurs at discrete time instants

when the state of the system reaches the learned boundary conditions.

In the learning algorithm, the switching between primitive controllers creates a dynamical system that has both continuous time and discrete time dynamics, thus making it a hybrid system. This notion of a hybrid system differs from other uses of hybrid systems, for example in [69]. Before describing this hybrid system, the dynamics and constraints of the system will be described in detail.

### 5.2.1 System Dynamics

The system dynamics under consideration in this chapter can be written as

$$\dot{x} = f(x, u) = \begin{cases} \dot{x}_I &= f_I(x_I, u) \\ \dot{x}_E &= f_E(x_E, x_I), \end{cases} \quad (189)$$

where the system state,  $x \in \mathbb{R}^n$ , is composed of two parts: an internal state  $x_I \in \mathbb{R}^{n_I}$  and an external state  $x_E \in \mathbb{R}^{n_E}$ . Thus,  $x := [x_I, x_E]^T$  and  $n = n_I + n_E$ . The internal state,  $x_I$ , describes the configuration of the system in reference to itself (e.g. actuator positions, joint angles, component velocities, etc.). The external state,  $x_E$ , describes the configuration of the system in reference to the outside world (e.g. location and velocity of the system in some global reference frame). An alternate way of describing these states is that the internal state needs proprioceptive sensors to measure its value while the external state needs exteroceptive sensors to measure its value.

It is assumed that the internal state is rectangularly bounded.<sup>1</sup> Let these bounds be described by  $x_{Imin} \in \mathbb{R}^{n_I}$  and  $x_{Imax} \in \mathbb{R}^{n_I}$ , where

$$x_{Imin}(j) \leq x_I(j) \leq x_{Imax}(j), \quad \forall j \in \{1, \dots, n_I\}.$$

In other words, each element in  $x_{Imin}$  is less than the corresponding element in  $x_{Imax}$ . The external state is allowed to be unbounded. The input to the system is given by  $u \in \mathbb{R}^m$ . For this class of systems, the input affects the internal state through  $f_I(x_I, u)$  while it only indirectly affects the external state through the coupling with  $x_I$ .

---

<sup>1</sup>This bound can be generalized to any polytopic boundary.



### 5.2.2 Primitive Controllers and Decision Conditions

It is assumed that the primitive controllers have been designed such that they will always move the internal state of the system until the state encounters a boundary<sup>2</sup>. In other words, the closed-loop system does not have any equilibrium points in the internal state space. Let  $\xi$  index the controller selection, the primitive controllers are defined as  $\kappa_\xi(x_I) : \mathbb{R}^{n_I} \rightarrow \mathbb{R}^m$ , which determine the input,  $u = \kappa_\xi(x_I)$ . Let the set of all primitive controllers be given by  $\mathcal{E} := \{1, \dots, k\}$  (with  $\xi \in \mathcal{E}$ ), where  $k$  is the number of different primitive controllers. Define the system dynamics while a particular controller is being applied as  $f_\xi(x) := f(x, \kappa_\xi)$ . Since,  $\kappa_\xi(x_I)$  can be any nonlinear function both the internal state and the external state can be controlled in arbitrary ways with the controller  $\kappa_\xi(x_I)$ .

Decisions are made on what control law to use when the internal state of the system intersects a decision boundary. Decision boundaries are used in the internal state space to greatly reduce the number of state-action pairs that are needed to decide when a new primitive controller needs to be applied. These decision boundaries are represented by  $n_I - 1$  dimensional hyperplanes in  $\mathbb{R}^{n_I}$ . Each hyperplane  $p_i$  is parameterized by two variables  $o_i \in \mathbb{R}^{n_I}$  and  $d_i \in \mathbb{R}^{n_I}$ , where  $o_i$  and  $d_i$  describe the origin and unit normal direction, respectively, of the  $i$ th hyperplane. Therefore, the hyperplanes are defined as  $p_i := \{x_I \mid (x_I - o_i)^\top d_i = 0\}$ . The set of all hyperplanes is  $P = \{p_1, \dots, p_\eta\}$ , where  $\eta$  is the total number of hyperplanes. The boundary that was last intersected by  $x_I$  is encoded with the boundary state variable  $\beta \in \mathcal{B}$ , where  $\mathcal{B} := \{0, 1, \dots, \eta\}$ . Initially, when the internal state has not yet intersected a boundary  $\beta = 0$ .

In order to ensure that the system can indeed learn how to locomote, constraints on the set of controllers and boundaries are needed. In particular, there must be a guarantee that a control law is always applicable. This means the system can always move away from a boundary once the boundary has been encountered. Also, it ensures the system will always

---

<sup>2</sup>The assumption is valid because this equates to the motion of low level motor controllers that are usually built into the hardware of a robotic system, which have limited operating range.

eventually encounter a boundary.

To establish this guarantee, it is assumed that the hyperplanes intersect to form a convex polytope. To describe this constraint more formally, let

$$\bar{D} := \{x_I \mid (x_I - o_i)^\top d_i < 0 \ \forall i \in \{1, \dots, \eta\}\}. \quad (190)$$

The set  $\bar{D}$  is the set of all points inside the polytope formed by the intersection of the hyperplanes in  $P$ . Thus, the constraint is that  $\bar{D}$  must be convex. This constraint also gives a minimum to the number of hyperplanes needed, i.e.,  $\eta_{min} = n_I + 1$ .

The set of primitive controllers move the internal state of the system around in the polytope defined by  $\bar{D}$ . A valid set of primitive controllers is a set such that, for each point along a given hyperplane, there is at least one primitive control action that moves the state away from that hyperplane and back into the convex polytope defined by  $\bar{D}$  for all hyperplanes in  $P$ . Thus, we assume that the boundary conditions and control laws have been designed such that

$$\forall i \in \{1, \dots, \eta\}, \exists j \in \mathbf{E} \mid s.t. \ d_i^\top f_j(x) < 0 \ \forall x \in p_i. \quad (191)$$

We also impose a non-transversality condition on the primitive controllers and the decision boundaries. This condition restricts the internal state to not move along the decision boundaries. Two important effects are caused by this condition. The first is that the internal state can not return to the same boundary without first encountering another boundary and the second is that the internal state can not stay in the interior of  $\bar{D}$  forever.

Verification of these effects can be seen by looking at the trajectories of the primitive controllers when initialized at different points along the boundaries. By continuity, these trajectories can never cross each other and, therefore, if a controller brings a state back to the same boundary then there must be a stationary, singular point on that boundary. This would violate the non-transversality condition in (191), thus the primitive controllers will never bring the internal state back to a boundary that it has just encountered. The second effect is verified by a similar reasoning as the first. For the internal state to stay

in the interior of the boundaries forever with the same primitive controller there must be a point along its trajectory that is tangential to the hyperplanes that make up the boundaries. Again, this violates the non-transversality condition in (191); and as result, the internal state will always eventually encounter a decision boundary.

In order to keep the notation simple it is assumed that the internal state will not encounter more than one decision boundary at a time. This assumption is reasonable because, for all but contrived systems, it is improbable for the internal state to intersect more than one hyperplane due to the non-transversality condition, which prevents the potentially non-pathological sliding along a boundary.

### 5.2.3 Hybrid System Formulation

Following the definition and notation in [16], the hybrid system is composed of four parts: (i) the flow map, which describes the continuous time evolution of the system; (ii) the flow set, which determines when the flow map takes place; (iii) the jump map, which describes the discrete time updates to the system; and (iv) the jump set, which determines when the jump map takes place. The interpretation is that the system flows during the continuous time evolution and jumps at the discrete time updates. Combine all the system information into a generalized state variable  $q := [x, \beta, \xi]^T \in \mathcal{Q}$ , where  $\mathcal{Q} = \mathbb{R}^n \times \mathcal{B} \times \mathcal{E}$ .

The flow set is defined with the bounds on  $x_I$ ,

$$C = \{q \in \mathcal{Q} \mid x_{I\min i} \leq q_i \leq x_{I\max i}, i \in \{1, \dots, n_I\}\}. \quad (192)$$

The jump set is thought as the complement to  $\bar{D}$ ,

$$D := \{q \in \mathcal{Q} \mid [I^{n_I \times n_I} \ 0^{n_I \times (n_E + 2)}]q \notin \bar{D}\}. \quad (193)$$

In words, (193) states that the jump set is the set of  $q$ 's where the first  $n_I$  components of  $q$  are not elements of  $\bar{D}$ . An illustration of an example internal state-space with boundaries, flow set, and jump set can be seen in Fig. 17.

Before defining the flow map and jump map for the system two other functions must first be introduced. The first is the boundary map,  $b(x_I) : \mathbb{R}^{n_I} \rightarrow \mathcal{B}$ , which maps the internal

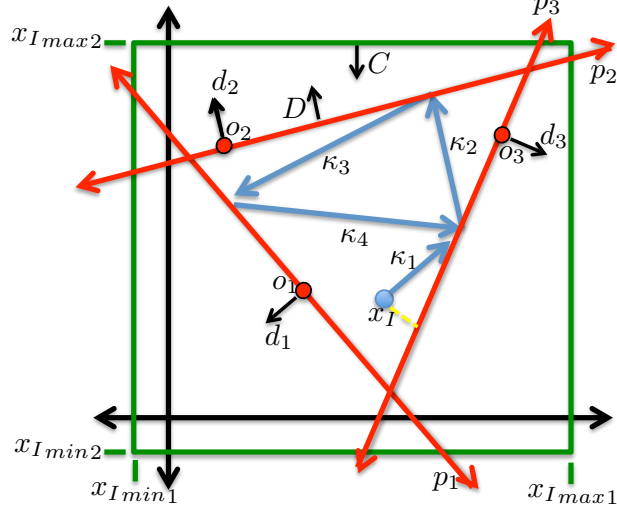


Figure 17: Illustration of the internal state  $x_I \in \mathbb{R}^2$  being controlled into a limit cycle with controllers  $\kappa_1, \dots, \kappa_4$  and with the location of the decision boundaries defined by the parameters  $o_i$  and  $d_i$ . The flow set  $C$  is the interior of the box and the jump set  $D$  is the exterior of the triangle formed by  $p_1$ ,  $p_2$ , and  $p_3$ . The interior of this triangle must be convex.

state to a boundary state. The second is the controller selector map,  $e(\beta) : \mathcal{B} \rightarrow \mathcal{E}$ , which selects which controller to use given a boundary state. The controller selector map for a given boundary must satisfy the condition in (191).

With the above functions the flow map and jump map of the hybrid system can be written as

$$f_{\mathcal{H}}(q, u) = \begin{bmatrix} f(x, u), & 0, & 0 \end{bmatrix}^T \quad (194)$$

$$g_{\mathcal{H}}(q) = \begin{bmatrix} x, & b(x_I), & e(b(x_I)) \end{bmatrix}^T \quad (195)$$

respectively. Thus,  $f_{\mathcal{H}}(q, u) : C \times \mathbb{R}^m \rightarrow \mathcal{Q}$  and  $g_{\mathcal{H}}(q) : D \rightarrow \mathcal{Q}$ . Finally, the hybrid system is defined as

$$\mathcal{H} : \begin{cases} \dot{q} = f_{\mathcal{H}}(q, u) & q \in C \\ q^+ \in g_{\mathcal{H}}(q) & q \in D \end{cases}. \quad (196)$$

### 5.2.4 Reward Function

Learning only makes sense if there is something to learn. To this end, a reward function and value function need to be associated to the system, which is usually determined trivially by what is desirable (e.g. if forward progress is desirable then distance forward is the reward or it can be as simple as a positive value for a reward and a negative value for a penalty as in a traditional reward function). Let the reward function be  $R(x_E, u) : \mathbb{R}^{n_E} \times \mathbb{R}^m \rightarrow \mathbb{R}$  and the corresponding value function becomes

$$V(x_E, u) = \int_{t_0}^{t_0+t_\pi} R(x_E, u) dt, \quad (197)$$

where  $t_0$  is the initial time and  $t_\pi$  is length of time that is being optimized over.

Given the above definitions, the objective is to maximize the value function,  $V(x_E, u)$ , without explicit knowledge of the system dynamics,  $f(x, u)$ , and the controllers,  $\kappa_i(x_I, \tau)$ , by learning the controller selector map,  $e(\beta)$ , and the set of boundaries,  $P$ . Two things to note are (i) the primitive controllers depend on  $x_I$  and not on  $x_E$  and (ii) that the reward and value functions depend on  $x_E$  and not  $x_I$ , which is why this is referred to a locomoting problem.

To relate the above framework to the example given about the robotic caterpillar in Section 5.1, the external state is the position of the caterpillar and the value function is how far forward it has moved. The internal states are the current configuration of its body. These internal states are bounded by how far the body can move back and forth, which defines the flow set. The primitive controllers could be to oscillate the body at different points or with different frequencies. Setting different decision boundaries with different jump maps will cause the body to move fast or slow and in or out of unison with other parts of the body. The goal is to find the setting that makes the robotic caterpillar move forward the “best” (as defined by the value function) by deciding when the given controllers are implemented.

### 5.3 Learning Algorithm

Learning the controller actions and the decision boundaries are the main focus of this section and *reinforcement learning* is primarily use to this end. This type of learning is often done as an online process, which adds the additional caveat that the agent must decide when it has sufficiently learned the environment and start utilizing its knowledge. As mentioned in Chapter 1, this is known as “exploration versus exploitation” [23]..

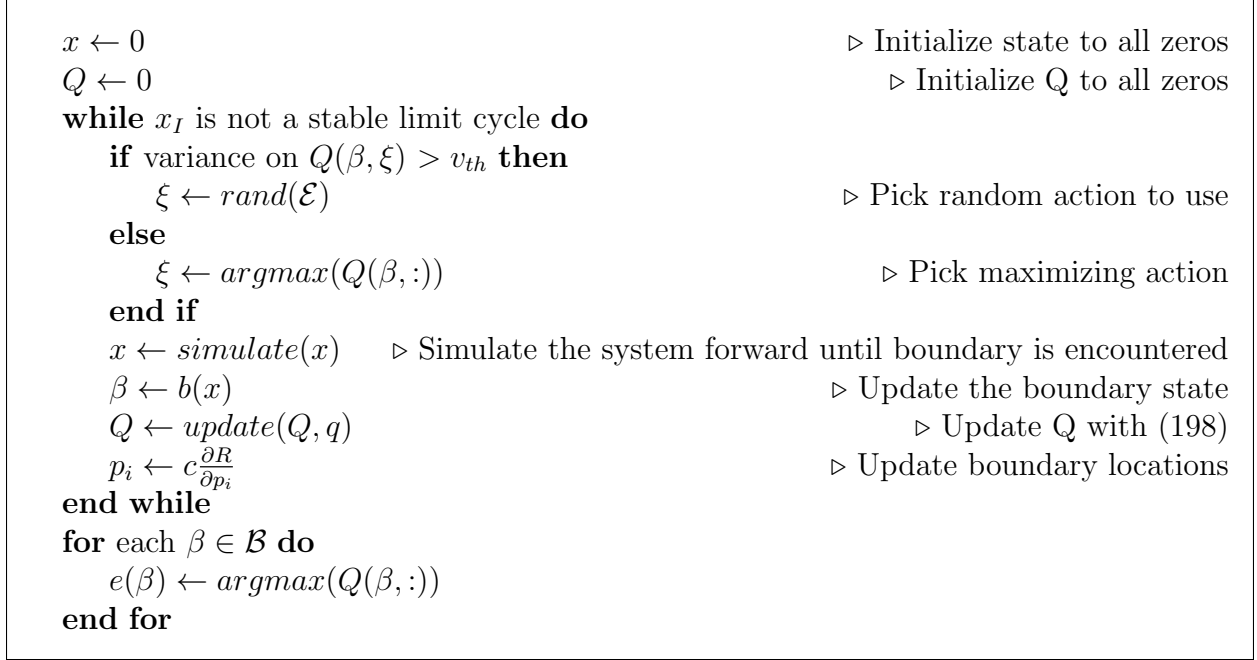
Reinforcement learning is usually modeled as a Markov Decision Process (MDP) [54] with four components:  $\mathcal{S}$ ,  $\mathcal{A}$ ,  $\mathcal{P}$ , and  $\mathcal{R}$ .  $\mathcal{S}$  is the set of states for the agent and the environment.  $\mathcal{A}$  is the set of actions or decisions that agent can take.  $\mathcal{P}$  is a function that defines the probabilities of transitioning from the current state to the next state given a certain action.  $\mathcal{R}$  is the function that determines the reward that is received after choosing a action from a given state.

With reinforcement learning the agent is attempting to learn an optimal policy,  $\pi$ , for the MDP, which is a description of how the agent chooses the actions to perform given a certain state. To do this the agents often learns the value function, which in turn will produce a policy. The value function,  $V$ , gives the maximum reward that can be earned from a given state. A variant of the value function is the value-action function,  $Q$ , which gives the maximum reward that can be earned from a given state after performing a given action. For the system framework presented in Section 5.2 the learning algorithm components are  $\mathcal{S} = \mathcal{B}$ ,  $\mathcal{A} = \mathcal{E}$ ,  $\mathcal{R} = R(x_E, u)$ , and  $\mathcal{P}$  is not explicitly used.

#### 5.3.1 Learning Controller Actions

A type of reinforcement learning known as Q-learning was one of the most important breakthroughs in the field of reinforcement learning [54]. Q-learning is an iterative update algorithm for the value-action function,  $Q$ ; hence the name. The value-action function is a variant of the value function, which gives the maximum reward that can be earned from a given state after performing a given action. A model that maps from actions to states

Figure 18: Learning To Locomote Algorithm



is not needed for either the learning or the action selection in Q-learning. For this reason, Q-learning is called a *model-free* method. This learning algorithm is guaranteed to converge to the optimal,  $Q^*$ , if all state-action pairs continue to be updated. Q-learning is used to learn controller actions given decision boundaries for the hybrid system defined in (196). The algorithm has a simple update rule,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( R_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right), \quad (198)$$

where  $s_t$  is the boundary state ( $\beta$ ) and  $a_t$  is the primitive controller ( $\xi$ ) at time  $t$ . In (198),  $\alpha$  is known as the learning rate and  $\gamma$  is known as the discount factor. Fundamentally, equation (198) states that the update  $Q$  for state  $s$  and action  $a$  will be the old  $Q$  for that pair plus a scaled sum of the instantaneous reward and the discounted maximum value that is currently known for the next state. A smaller  $\alpha$  means old information will be trusted more than new information. And a smaller  $\gamma$  means instantaneous rewards are more important than future rewards. From (198) a policy,  $\pi$ , is generated from  $Q$  in a “greedy” manner. This the action that is selected for a given state is the one that maximizes the  $Q$  value for that state.

### 5.3.2 Learning Decision Boundaries

A different approach is used to find the optimal decision boundary locations. A gradient ascent algorithm is used to iteratively move the boundaries to the optimal locations. The boundaries are moved proportional to the positive of the gradient of the value function with respect to the boundary locations,  $\frac{\partial V}{\partial p_i}$ . This proportion is set by the parameter  $c$ .

To estimate the gradient, each time the internal state  $x_I$  reaches one of the boundaries in  $P$  the position of that boundary is randomly changed by some small amount,  $\Delta p$ . The change in the reward function,  $\Delta R$ , is calculated over this change in the boundary position. The ratio of  $\Delta R$  to  $\Delta p$  is used as an approximation for the gradient  $\frac{\partial V}{\partial p_i}$ . The boundaries are moved by the amount equal to  $c \frac{\Delta R}{\Delta p}$ . This results in the boundaries moving in a “greedy” direction; in other words, a direction that maximizes the short term reward not necessarily the long term value.

The orientation of the boundaries,  $d_i$ , are currently chosen in one of two ways. The boundary orientations are chosen to either align with the axes of the state space or chosen randomly but with the constraint that the interior of the boundaries form a convex polytope.

The number of decision boundaries,  $\eta$ , can be no lower than  $\eta_{min} = n_i + 1$ , but there is no upper limit on  $\eta$ . As  $\eta$  increases a larger number of decisions are made for each cycle of the interior state, thus the better the results as is shown in the results in Section 5.4.1. The improved results come at cost in convergence times, which scale as  $O(|\mathcal{B}| \sum_{\beta \in \mathcal{B}} |\mathcal{E}(\beta)|) = O(\eta^2 k)$  [27].

### 5.3.3 Exploration vs. Exploitation

Deciding when the agents have learned sufficient information and deciding when to begin executing the learned policy is a current area of research in reinforcement learning. Exploration and exploitation is done by knowing that the learning process is Q-learning, which guarantees that the  $Q$  values will converge to the optimal value if the state-action pairs continue to be updated. Thus, the variance in the  $Q$  values will converge to zero. The variance



in the  $Q$  values is used to determine when the learning algorithm should explore or exploit.

Exploration strategies are usually grouped into two categories: *undirected* and *directed* [58]. Undirected techniques use no knowledge of the learning process and ensure exploration by merging randomness into the action selection. Directed techniques utilize knowledge of the learning process to preform exploration in more directed manner.

Exploitation takes place when the maximum variance in the last  $\sigma_n^2$  updates of  $Q$  for a particular state,  $s$ , is below a threshold of  $\sigma_{th}^2$ . Otherwise exploration is performed. This variance for a state is denoted as  $\sigma^2(s)$ . During exploration actions are picked randomly with a distribution that is proportional to  $\sigma^2(s)$ . If a state-action pair in  $Q$  has not been updated more that  $\sigma_n^2$  times the variance is set to a large number,  $\sigma_{inf}^2$ . This method assures that exploitation will not take place until each state-action pair has been attempted  $\sigma_n^2$  times and that the variance on the estimated  $Q$  values for each state-action pair is below  $\sigma_{th}^2$ .

## 5.4 Examples

The efficacy of the learning algorithm from Section 5.3 is demonstrate on three systems in this section. Two system are simulated systems and one is a real robotic system.

### 5.4.1 Example Simulated System

We demonstrate the ability of our learning algorithm on the nonholonomic integrator [5] known as “Brockett’s system” [47]. Brockett’s system is an ideal example system to test and demonstrate the learning algorithm outlined in Section 5.3 because it is one of the simplest systems that fits the model defined in Section 5.2. In addition, due to the so-called topological obstruction there is no continuous control law to stabilize Brockett’s system [47]. The dynamics of Brockett’s system are

$$f(x) = [u_1, u_2, x_{I1}u_2 - x_{I2}u_1]^T, \quad (199)$$

where  $u \in \mathbb{R}^2$ ,  $x_I \in \mathbb{R}^2$ , and  $x_E \in \mathbb{R}$ . Brockett’s system is a surprisingly rich system given its innocuous appearance.

For this system, we define the primitive controllers such that they move the internal state,  $x_I$ , with unit magnitude. The direction for each controller is random and is drawn from eight uniform random distributions. The domain of these distributions are each equal to one-eighth partitions of the unit circle, which guarantees that the condition in (191) is satisfied. We use 32 controllers and set the bounds on  $x_I$  as  $x_{I_{min}} = [-1, -1]^T$  and  $x_{I_{max}} = [1, 1]^T$ .

Lastly, we select four boundaries ( $\eta = 4$ ) for the learning algorithm. The directions of the boundaries are fixed to  $d_1 = 0$ ,  $d_2 = \pi/2$ ,  $d_3 = \pi$  and  $d_4 = 3\pi/2$ . The origin's of the boundaries,  $o_i$ , are chosen randomly such the constraint that  $\bar{D}$  is convex is satisfied.

From (199) it is seen that for this example system  $m = 2$ ,  $n_I = 2$ ,  $n_E = 1$ ,  $n = 3$ ,  $\eta = 4$ , and  $Q$  is a  $4 \times 32$  matrix. The reward function is defined as  $R(x_E, u) = \frac{dx_E}{dt}$ . The parameters used for the learning algorithm were found empirically and are as follows:  $\alpha = 0.75$ ,  $\gamma = 0.25$ ,  $c = 0.1$ ,  $\sigma_n^2 = 3$ ,  $\sigma_{th}^2 = 0.025^2$ , and  $\sigma_{inf}^2 = 10^2$ . To simulate the system the algorithm shown in Fig. 18 is executed in Matlab.

Using this learning algorithm we were able to learn the optimal control sequence and boundary locations given the setup described above. Results of the learning algorithm are shown in Fig. 19 and explained further in Section 5.4.2. It is known that the optimal continuous controller for Brockett's system is sinusoidal of the form

$$\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} \cos(\lambda t) & -\sin(\lambda t) \\ \sin(\lambda t) & \cos(\lambda t) \end{bmatrix} \begin{bmatrix} u_1(0) \\ u_2(0) \end{bmatrix}, \quad (200)$$

where  $\lambda$  and  $u(0)$  can be solved for given initial and desired final states of the system [47]. We compared the external state value after running both the optimal control law in (200) and the control law learned with our learning algorithm. The results after running 100 trials for two different cases are shown in Table 4. These results for varying  $\eta$  and  $k$  in this way agree with the theory shown in Section 5.3.2.

#### 5.4.2 Explanation of Results For Simulated System

The results of an experiment using the learning algorithm (described in Section 5.3) on the Brockett's system (described in Section 5.4.1) are shown in Fig. 19. The top plot shows the

Table 4: External state with different parameters using 100 trials of our learning compared against the optimal state value.

$\eta$	$k$	avg. # iterations	avg. % of optimal
4	32	420	90.3%( $\pm 3.8\%$ )
8	64	3377	96.1%( $\pm 2.7\%$ )

internal state components,  $x_{I1}$  and  $x_{I2}$ , as well as the state constraints. The second plot shows the external state  $x_E$  and the value of the reward function. The third plot shows the controller state  $\xi$  and boundary state  $\beta$ . And the bottom plot shows the maximum variance in the  $Q$  values,  $\max(\sigma^2(s))$ , as well as the threshold value for when exploitation takes place,  $\sigma_{th}^2$ . The times when the learning algorithm is exploring and exploiting are shown in each of the four plots. The boundary locations are implicitly shown in the top plot by the envelope of the internal states.

In this experiment the states are all initialized to zero. It can be seen that during the first portion of this experiment (time 0 to 220) the learning algorithm is only exploring. During exploration the control state is random and the boundaries locations are moving but not in a consistent direction. In addition, the external state and reward value have an average output of zero. The first time exploitation takes place is when the time approximately equals 220, which can be seen in the bottom plot because  $\max(\sigma^2(s))$  falls below  $\sigma_{th}^2$ . Exploration and exploitation trade off for the middle portion of the experiment (time 220 to 420). In the last portion of the experiment only exploitation takes place (time 420 to 600). The control state and boundary state settle into a repeating pattern and the value of  $x_E$  quickly increases. Note that even after the time when only exploitation is taking place (approx. time of 420) the boundaries are still moving. The boundaries are moving towards positions that give maximum reward. The boundaries settle at the limits of  $x_I$  and the average reward value stops increasing.

### 5.4.3 Simulated High Dimensional System

Another simulated system was constructed to demonstrate the learning algorithm's capabilities on a system of high dimension. The dynamics of this system were created to resemble that of an  $N$ -jointed serpent swimming through water (see Fig. 20). The serpent is constrained to move in the  $xy$ -plane. Its "head" is at the position  $(x, y)$  and it is always oriented along the  $x$ -axis. The serpent has  $N$  movable body parts or links. The  $i$ th link is of length  $l_i$  and begins with a 1-dimensional rotational joint with angle  $\theta_i$  and angular velocity  $\dot{\theta}_i$ . It has a mass,  $m_i$ , at the center of the link, and a fin with an associated constant  $c_i$ . Rotating the joint causes a force,  $f_i$ , to be applied to the mass that is proportional to  $c_i$  and the square of the angular velocity of joint,  $\dot{\theta}_i^2$ . The direction of the force  $f_i$  is the negative of the direction of the linear velocity of the mass  $m_i$ . The acceleration of the serpent is equal to the sum of the forces divided by the sum of the masses. The serpent has direct control of the angular velocity of each of its  $N$  joints. Thus, the input,  $u$ , to the system is an element of  $\mathbb{R}^N$ .

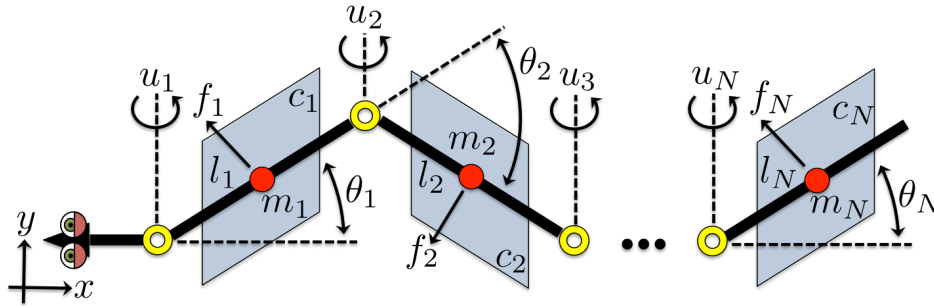


Figure 20: An illustration of the  $N$ -link high dimensional simulate serpent system.

The goal of the serpent is to learn to move its joints in such a way that it moves as quickly as possible in the  $x$ -direction with no concern for its movement in the  $y$ -direction. The serpent has set of  $k$  primitive controllers to use that it innately knows and can use accomplish this goal.

Since the serpent resembles a serial link robotic manipulator the kinematics of serpent

are solved in a similar fashion to that of an  $N$ -link robotic manipulator [51]. In other words, given the serpent's current joint angles and angular velocities the positions,  $p$ , and velocities,  $\dot{p}$ , of the masses of all the links can be computed. To compute the positions and velocities of the masses given the current state of the serpent let us define the forward kinematics function  $FK(x, \dot{x}, y, \dot{y}, \theta, u) = [p, \dot{p}]$ .

Given the forward kinematics the dynamics of the serpent can be written as

$$\ddot{x} = \frac{\sum_{i=1}^N -c_i \dot{p}_{xi}^2}{\sum_{i=1}^N m_i}, \quad \ddot{y} = \frac{\sum_{i=1}^N -c_i \dot{p}_{yi}^2}{\sum_{i=1}^N m_i}, \quad (201)$$

where  $\dot{p}_{xi}$  and  $\dot{p}_{yi}$  are the speed of  $m_i$  in the  $x$ -direction and  $y$ -direction, respectively. From this the internal state is defined as  $x_I = \theta$ , the external state is defined as  $x_E = [x, y]^T$  (with some abuse of notation for  $x$ ).

The boundaries for the learning algorithm have fixed orientations such that they form a “hyper rectangular prism” in the internal state space. This means there are  $2N$  boundaries. The  $k$  primitive controllers for the serpent are chosen randomly from elements in  $\mathbb{R}^N$ , with the constraint that there is always at least one controller with a positive sign and another controller with a negative sign for each of the  $N$  elements. This guarantees that the non-transversality condition in (191) is satisfied. It also means  $k \geq 2N$ .

Relating the serpent dynamics of (201) to the hybrid system of (196) we have  $m = N$ ,  $n_I = N$ ,  $n_E = 2$ ,  $n = 2N + 2$ ,  $\eta = 2N$ , and  $k \geq \eta$ . The bounds on  $x_I$  as  $x_{Imin} = [-\pi, \dots, -\pi]^T$  and  $x_{Imax} = [\pi, \dots, \pi]^T$  and reward function is defined as  $R(x_E, u) = \dot{x}$ .

Lastly, the authors are well aware that the dynamics as they are written in (201) are not an accurate representation of the actual dynamics for a  $N$ -link swimming serpent. The contribution of this section is not to present high fidelity serpent dynamics but to demonstrate the capabilities of the learning algorithm on a high dimensional system.

The learning algorithm was applied to a 15-link serpent system with 315 primitive controllers. It would be infeasible to discretize the internal state space of this system and learn state-action pairs for each possible combination. Even just discretizing each internal state dimension into 2 parts would create over 1 million state-action pairs to test and learn. With

our learning algorithm using 30 boundaries for this system creates 9450 boundary-action pairs to test and learn. The results in Fig. 21 show that the system reaches a steady limit cycle with the external state increasing at a steady maximum rate after approximately  $8 \times 10^4$  updates.

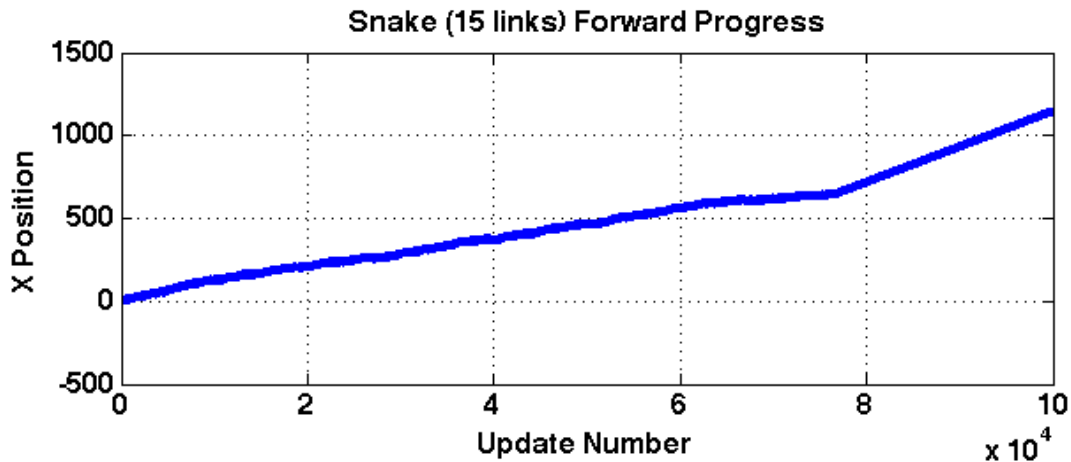


Figure 21: Forward movement verse update number of a 15-link serpent simulated system learning to move forward. After  $8 \times 10^4$  updates the system has entered a stable limit cycle and moves forward at the maximum rate.

#### 5.4.4 Physical Robotic System

In addition to running our learning algorithm on a simulated system, we tested the algorithm on a physical robotic system. A photograph of this robot is shown in Fig. 16. The fabrication of this robot was for the sole purpose of testing the learning algorithm. This robot consists of a body and two movable appendages. Each appendage has only one rotational degree of freedom. The body of the robot holds an Arduino microcontroller, two servos, ultrasonic distance sensor, XBee wireless transmitter, and two 9V batteries. To maintain a low cost and rapid construction time the entire structure of the robot is made out of cardboard, which is held together with hot glue and two small cabinet hinges. The learning algorithm does not run directly on the hardware of the robot but runs on a separate computer and communicates with the robot over a wireless link. The computer sends actuation commands

wirelessly to the robot and the robot sends servo positions and distance readings back to the computer.

This robot is limited to moving along a straight line and the objective of the robot is to move as far along that line as possible. It is impossible for the appendages of the robot to lose contact with the ground, thus for the robot to move it must “scoot”. Locomotion is only possible with differences in frictional forces from the two appendages and the ground. This complexity makes conceptualizing the necessary sequence of movements for forward progress difficult and not intuitive.

Instead of attempting to design the sequence of actuations for the robot, it learns them with our learning algorithm. The internal state,  $x_I$ , are the positions of its two appendages and the external state,  $x_E$ , is the distance from the origin. The primitive controllers are all possible combinations of moving the appendages up, down, and not at all. However, the case when both are not moving is not included as a primitive controller. This makes eight possibilities, therefore  $k = 8$ . The boundaries are the same as in the simulated example (Section 5.4.1), thus  $\eta = 4$ .

Using the learning algorithm presented in Section 5.3, the robot is able to learn the necessary movements to move forward and backward. After running the learning algorithm several times, we observe that the robot learns to move forward by positioning the forward appendage at a large angle relative to the ground and proceeding to move the back appendage back-and-forth. This causes the robot to scoot forward at approximately 1 cm/s. To move backward, the robot lays its front appendage down flat and uses the back appendage to pull itself backward. It moves backwards at roughly 0.25 cm/s. Plots of the robot’s movements are shown in Fig. 22 <sup>3</sup>.

---

<sup>3</sup>A movie of the robot learning can be viewed at <http://gritslab.gatech.edu/home/2012/11/learning-to-locomote/>.

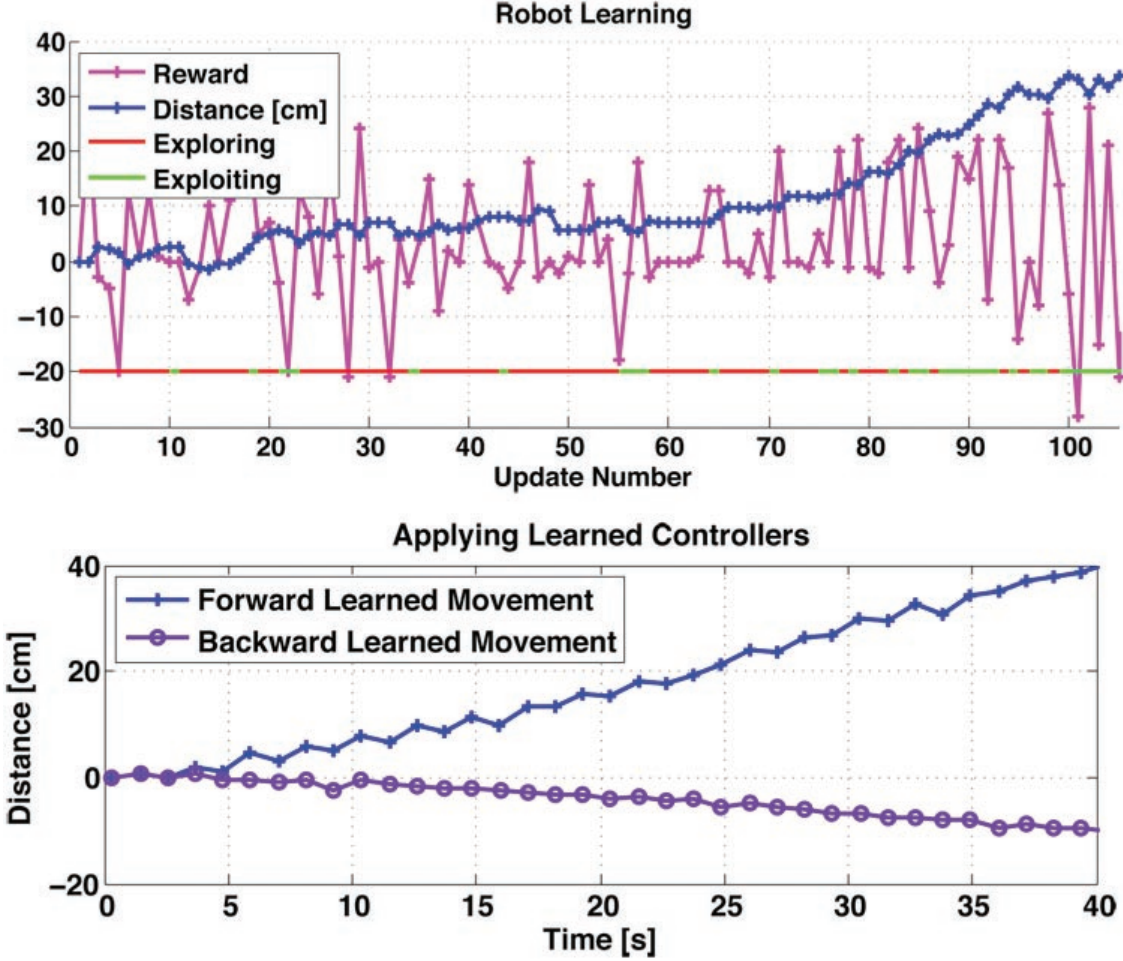


Figure 22: (Top) Plot shows the physical robot learning to move forward. The forward progress is shown (blue (dark) line with plus sign markers) as well as when the robot is exploring (red (dark) marks) and exploiting (green (light) marks). (Bottom) Plot shows the robots movements after learning how to move both forward (line with plus sign markers) and backwards (line with circle markers).

## 5.5 Conclusion

This chapter has introduced a new algorithm to learn how to maximize an objective function of a complex locomoting system by learning when to switch between primitive controllers. The system is described as a hybrid system because it contains both continuous time and



discrete time dynamics due to the switching between primitive controller actions. The hybrid system formulation allows for a concise representation of the system and the switching boundaries. The primitive controllers act directly on what is referred to as the internal state and the objective function is based only on what is referred to as the external state. The switching boundaries and the primitive controllers are used as state-action pairs for a Q-learning algorithm, which learns the controller to apply at each boundary encountered by the internal state. The locations of the boundaries are adjusted using a gradient ascent algorithm. The capacity of this learning algorithm is demonstrated on both a simulated system with a known analytical solution, a simulated system of high dimensionality, and on a real robot with complex dynamics due to friction. By learning boundary-controller pairs to maximize the objective function this algorithm mitigates the “curse of dimensionality”.

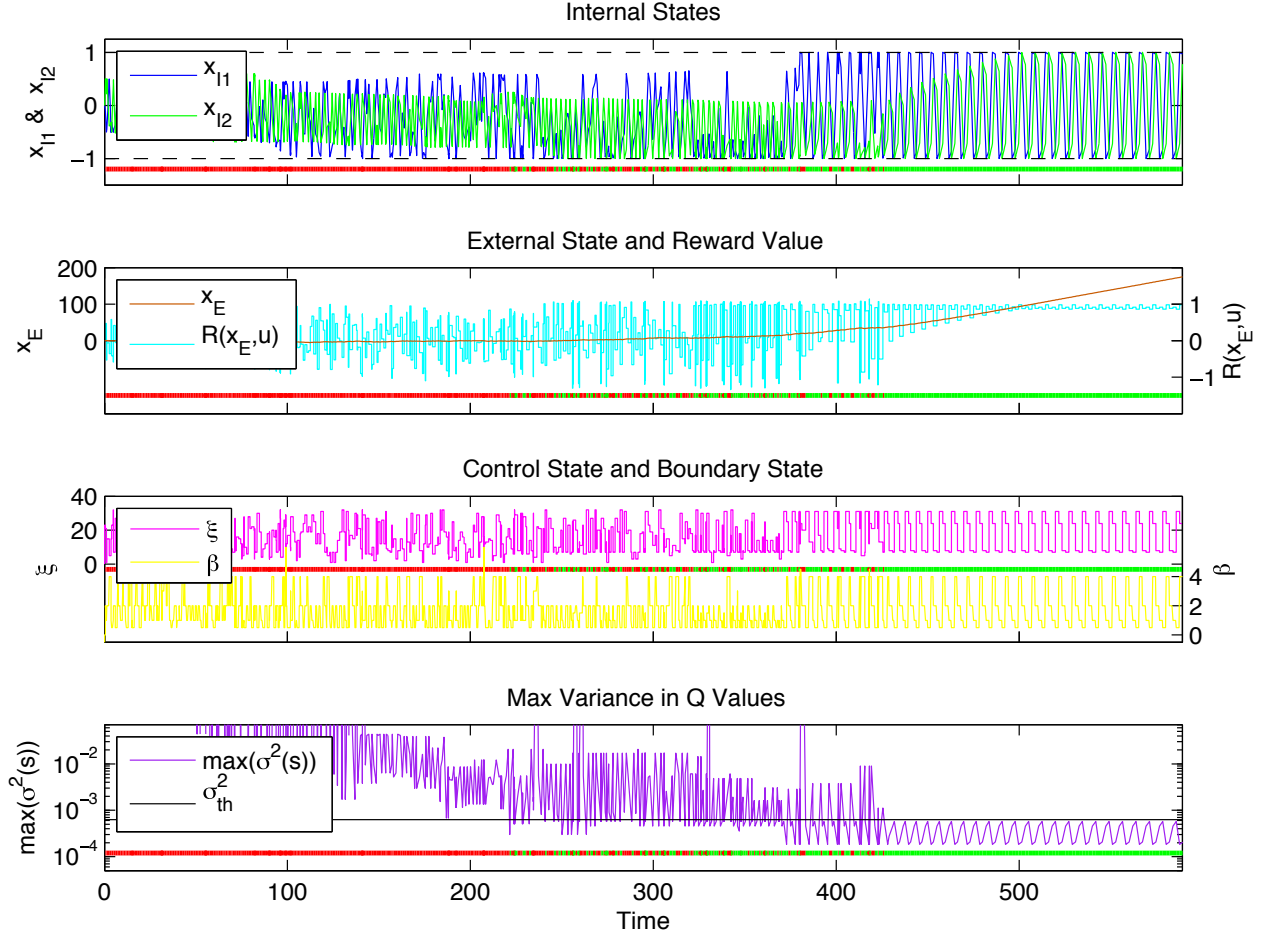


Figure 19: These plots show the results of running the learning algorithm on Brockett's system. Top: Plot of the internal state components,  $x_{I1}$  (blue (dark) line) and  $x_{I2}$  (green (light) line), verses time. The internal state constraints are also shown (black dashed lines). Second: Plot of the external state and reward,  $x_E$  (orange (dark) line) and  $R(x_E, u)$  (blue (light) line), respectively, verses time. Third: Plot of the control state and boundary state,  $\xi$  (top pink (dark) line) and  $\beta$  (bottom yellow (light) line), respectively, verses time. Bottom: plot of the maximum variance in the Q values for a given state,  $\max(\sigma^2(s))$  (purple (dark) line), verses time. The variance threshold for deciding between exploration and exploitation is also shown (straight solid black line). In all the plots it is shown when the learning algorithm is exploring or exploiting with the red (dark) and green (light) marks, respectively, across the bottom of each plot.

## CHAPTER VI

### CONCLUSION

The 18th century offered major innovations in materials, the 19th century offered major innovations in industrial machines, and the 20th century offered major innovations in electronics. The 21st century will make major innovations in *robotics* as the frequency of robot-people interactions grow exponentially. This growth will emulate Moores law of the 20th century that describes the growth rate of personal computers. For robots to achieve this level of human interaction, Rodney Brooks has said robots must have a 2 year olds ability to recognize objects, a 4 year olds ability to understand language, a 6 year olds ability to manipulate objects, and an 8 year olds ability to understanding social behavior [7]. For a robot to have the capacity to perform these tasks at the levels of a human child, significant research still needs to be done in the areas of machine learning and control theory. The research covered in this work makes advancements in these two areas of research and moves robots closer to having the abilities to be more autonomous and make better decisions on their own.

This research began by defining the *Ergodic Environmental Exploration* ( $E^3$ ) algorithm, which is used to optimally explore unknown environments with areas of varying degrees of importance. The  $E^3$  algorithm generates a trajectory for a dynamical system to discover and collect information from an unknown underlying information distribution in the exploration space of interest. The algorithm minimizes control effort and the ergodic cost of the trajectory; the difference between the time average behavior of the trajectory to the spatial distribution of maximum information gain.

Following the theoretical introduction of the  $E^3$  algorithm, experimental results of an agent using the algorithm to learn an underlying information distribution to optimally

explore an unknown space were presented. These experiments showed successful employment of the  $E^3$  algorithm on real robots to explore a space in the ground and in the air, demonstrating the utility of the algorithm. To fully realize the impact of this research, it is important to optimize the  $E^3$  for computational speed and run validation experiments in higher dimensions. In doing so, roboticists will be able use this research on more complex systems in real-time.

To determine if a robot can even accomplish a given learning task, this research considered the conditions under which a given dynamical system could succeed at a given learning objective. This research defined the “learnability” of a dynamical system, which defines when a dynamical system can and can not learn. Given this definition, a theorem and proof were provided on when a linear system with a quadratic cost is “learnable”. Following this theorem, it becomes clear that the ground robot used in the experiments for optimal exploration would not succeed if it had to explore a 3 dimensional volume.

Finally, this research introduced a new algorithm to learn how to maximize an objective function of a complex locomoting system by learning when to switch between primitive controllers. The system is described as a hybrid system because it contains both continuous time and discrete time dynamics due to the switching between primitive controller actions. The capacity of this learning algorithm was demonstrated on a simulated system with a known analytical solution, a simulated system of high dimensionality, and on a real robot with complex dynamics due to friction. By learning boundary-controller pairs to maximize the objective function this algorithm mitigates the “curse of dimensionality”.

The overall thread that ties the different aspects of this research together is that by looking at learning problems from a control theoretic point of view, reveals new information derived from the system dynamics, which helps direct and focus the learning process. In conclusion, this research moves the field of robotics closer to autonomy, by advancing the underlying tools and employing new applications in the fields of control theory and machine learning.

# APPENDIX A

## NOTATION

### Notation General Notes:

- Left-hand subscript indicate epoch iteration number. Example  ${}_i\phi(x)$  is the  $i$ th value of  $\phi(x)$ .
- Generally an uppercase variable is the domain set for corresponding lowercase variable. Example  $X$  is the domain set for  $x$ .
- A variable with a hat is a the estimate of the variable without the hat. Example  $\hat{\Phi}$  and  $\Phi$ .

### Time variables:

$t_0$ : Initial time.

$t_f$ : Final time.

$T$ : Time domain,  $T = [t_0, t_f]$ .

$t_D$ : Time duration,  $t_D = |T| = t_f - t_0$ .

$t$ : An arbitrary point in time,  $t \in T$ .

$t_\Delta$ : Time sample step size,  $t_\Delta$  chosen s.t.  $\frac{t_f - t_0}{t_\Delta} \in \mathbb{N}$ .

$N_t$ : Number discrete time indices or size of discrete time domain set,  $N_t \in \mathbb{N}$ .

$K_t$ : Discrete time index domain set,  $K_t = \{0, 1, \dots, N_t - 1\}$ .

$k_t$ : An arbitrary value in the time index domain,  $k_t \in K_t$ .

$T_s$ : Discrete time domain set,  $T_s = \{t_0, t_0 + t_\Delta, \dots, t_0 + (N_t - 1)t_\Delta\}$ .

$t_s$ : An arbitrary point in discrete time,  $t_s \in T_s$ .

### **Spacial state variables:**

$n$ : State space dimensionality.

$x_m$ : Minimum state value vector,  $x_m \in \mathbb{R}^n$ .

$x_M$ : Maximum state value vector,  $x_M \in \mathbb{R}^n$ .

$x_D$ : State domain size vector,  $x_D \in \mathbb{R}^n$  and  $x_{D_i} = x_{M_i} - x_{m_i}$ .

$X$ : Spacial state domain. Rectangular region in  $\mathbb{R}^n$  defined as  $[x_{m_1}, x_{M_1}] \times \dots \times [x_{m_n}, x_{M_n}] \subset \mathbb{R}^n$ .

$x$ : An arbitrary point in spatial state space,  $x \in X$ .

$x_i$ : Represents the  $i$ th component of  $x$ ,  $x = [x_1, \dots, x_n]^T$ .

$\mathcal{X}$ : State spatial trajectory domain,  $\mathcal{X} \subset \mathbb{L}_2^n[T]$ .<sup>1</sup>

$x(t)$ : The state location at time  $t$ ,  $x(t) \in X$ .

$x(T)$ : The state spatial trajectory over all  $T$ ,  $x(T) \in \mathcal{X}$ .

$x[k_t]$ : The state location at time index  $k_t$ ,  $x[k_t] = x(k_t t_\Delta)$ .

$x_\Delta$ : State quantize step size vector,  $x_\Delta \in \mathbb{R}_+^n$  and chosen s.t.  $\frac{(x_{M_i} - x_{m_i})}{x_{\Delta_i}} \in \mathbb{N}$  for  $i \in \{1, \dots, n\}$ .

$N_x$ : Number quantized state indices in each component of  $x$ ,  $N_x \in \mathbb{N}^n$ .

$K_x$ : Discrete state index domain set,  $K_x = \{1, 2, \dots, N_{x_1}\} \times \dots \times \{1, 2, \dots, N_{x_n}\} \subset \mathbb{N}^n$ .

---

<sup>1</sup> $\mathbb{L}_p^n[D]$  represents the space of Lebesgue integrable functions of  $n$ th-dimension with  $p$ th-power over the domain  $D$ .

$k_x$ : An arbitrary value in the state index domain,  $k_x \in K_x$ .

$X_s$ : Quantized state domain set,  $X_s = \{x_{m_1}, x_{m_1} + x_{\Delta_1}, \dots, x_{m_1} + (N_{x_1} - 1)x_{\Delta_1}\} \times \dots \times \{x_{m_n}, x_{m_n} + x_{\Delta_n}, \dots, x_{m_n} + (N_{x_n} - 1)x_{\Delta_n}\}$

$x_s$ : An arbitrary quantized point in state space,  $x_s \in X_s$ .

### **Input variables:**

$m$ : Input space dimensionality.

$u_m$ : Minimum input value vector,  $u_m \in \mathbb{R}^m$ .

$u_M$ : Maximum input value vector,  $u_M \in \mathbb{R}^m$ .

$U$ : Input spatial domain. Rectangular region in  $\mathbb{R}^m$  defined as  $[u_{m_1}, u_{M_1}] \times \dots \times [u_{m_m}, u_{M_m}] \subset \mathbb{R}^m$

$u$ : An arbitrary point in input space,  $u \in U$ .

$\mathcal{U}$ : Input spatial trajectory domain,  $\mathcal{U} \subset \mathbb{L}_2^m[T]$ .

$u(t)$ : The input location at time  $t$ ,  $u(t) \in U$ .

$u(T)$ : The input spatial trajectory over all  $T$ ,  $u(T) \in \mathcal{U}$ .

### **Dynamics variables:**

$f(x, u)$ : State dynamics,  $f : X \times U \rightarrow \mathbb{R}^n$  and  $\dot{x} = f(x, u)$ .

$A(t)$ : Linearized state dynamics with respect to the state,  $A(t) \in \mathbb{R}^{n \times n}$  and  $A(t) = \frac{\partial f(x(t), u(t))}{\partial x}$ .

$B(t)$ : Linearized state dynamics with respect to the input,  $B(t) \in \mathbb{R}^{n \times m}$  and  $B(t) = \frac{\partial f(x(t), u(t))}{\partial u}$ .

### **Tuple (State and Input) variables:**

$\Sigma$ : State and input (feasible or infeasible) trajectories domain,  $\Sigma \subset \mathbb{L}_2^{n \times m}[T]$ .

$\sigma(T)$ : Tuple of (feasible or infeasible) state and input trajectories,  $\sigma(T) = (\chi(T), \mu(T))$  where  $\chi(T) \in \mathcal{X}$  and  $\mu(T) \in \mathcal{U}$ , and  $\sigma(T) \in \Sigma$ .

$\mathcal{Z}$ : State and input feasible trajectories domain,  $\mathcal{Z} \subset \Sigma$ .

$z(T)$ : Tuple of feasible state and input trajectories,  $z(T) = (x(T), u(T))$  where  $x(T) \in \mathcal{X}$  and  $u(T) \in \mathcal{U}$ , and  $z \in \mathcal{Z}$ .

### Spectral state variables:

$N$ : Vector of the number of spectral state coefficients in each dimension of the state.  
 $N \in \mathbb{N}^n$ .

$K$ : Spectral state coefficient index domain.  $K = \{0, 1, \dots, N_1-1\} \times \dots \times \{0, 1, \dots, N_n-1\}$ .

$k$ : Spectral state coefficient index.  $k \in K$ .

$\mathcal{E}$ : Spectral state coefficient domain,  $\mathcal{E} \subset \mathbb{R}^n$ .

$\xi_k$ : An arbitrary point in spatial frequency space (i.e wavenumber),  $\xi_k \in \mathcal{E}$  and  $\xi_k = \frac{k}{x_D}$  (element-wise inverse of  $x_D$ ).

$f_k(x)$ :  $k$ th Fourier basis function,  $f_k : X \rightarrow \mathbb{C}$  and  $f_k(x) = e^{2\pi i \xi_k^\top x}$ .

$F_k(x(T))$ :  $k$ th Fourier coefficient of the state trajectory,  $F_k : \mathcal{X} \rightarrow \mathbb{C}$  and  $F_k(x(T)) = \frac{1}{|T|} \int_T \overline{f_k(x(t))} dt$ .<sup>2</sup>

### Spatial distribution variables

$\phi(x)$ : Spatial distribution value at location  $x$ ,  $\phi : X \rightarrow \mathbb{R}_+$  and  $\int_X \phi(x) dx = 1$ .

$\phi(X)$ : Spatial distribution function on the domain  $X$ ,  $\phi(X) \in \mathbb{S}(\mathbb{R}^n)$ .<sup>3</sup>

$\Phi_k$ :  $k$ th spatial distribution spectral component,  $\Phi_k \in \mathbb{C}$  and  $\Phi_k(\phi(X)) = \int_X \overline{f_k(x)} \phi(x) dx = \mathbf{E} \left[ \overline{f_k(x)} \right]_{\phi(x)}$ .

---

<sup>2</sup> $\overline{f(x)}$  represent the complex conjugate of the function  $f(x)$ .

<sup>3</sup> $\mathbb{S}(\mathbb{R}^n)$  represents the space of Schwartz functions on  $\mathbb{R}^n$  (i.e. space of rapidly decreasing functions on  $\mathbb{R}^n$ ).



### Spatial distribution estimation variables:

$s^2$ : Spatial distribution measurement noise variance,  $s^2 \in \mathbb{R}$ .

$w$ : Spatial distribution measurement noise,  $w \sim N(0, s^2)$ .

$\tilde{y}(x)$ : Spatial distribution measurement model,  $\tilde{y} \in \mathbb{R}_+$  and  $\tilde{y}(x) = \max(\phi(x) + w, 0)$ .

$y$ : Spatial distribution measurement,  $y \in \mathbb{R}_+$  and  $y(x) = \max(\phi(x) + w, 0)$ .

$y(t)$ : Spatial distribution measurement at time  $t$ ,  $y(t) \in \mathbb{R}_+$ .

$y[k_t]$ : Spatial distribution measurement at time index,  $k_t$ ,  $y[k_t] = y(k_t t_\Delta)$ .

$y[K_t]$ : Spatial distribution measurement series for all indices in  $K_t$ ,  $y([K_t]) \in \mathbb{R}_+^{N_t}$ .

$\hat{\Phi}_k$ :  $\hat{\Phi}_k(x[K_t], y[K_t])$ ,  $k$ th Fourier coefficient of measurement series,  $\Phi_k \in \mathbb{C}$ ,  $\hat{\Phi}_k(x[K_t], y[K_t]) \approx \Phi_k(\phi(X))$ , and

$$\hat{\Phi}_k(x[K_t], y[K_t]) = \sum_{k_t \in K_t} y[k_t] \overline{f_k(x[k_t])}.$$

$\hat{y}(x)$ : Estimate of unscaled spatial distribution measurement at location  $x$ ,  $\hat{y} : X \rightarrow \mathbb{R}_+$  and

$$\hat{y}(x) = \left| \sum_{k \in K} \hat{\Phi}_k f_k(x) \right|.$$

$c(x)$ : Normalizing function,  $c : X \rightarrow \mathbb{R}$ .

$\hat{\phi}(x)$ : Estimate of spatial distribution value at location  $x$ ,  $\hat{\phi} : X \rightarrow \mathbb{R}_+$ ,  $\int_X \hat{\phi}(x) dx = 1$ , and

$$\hat{\phi}(x) = c(x) \hat{y}(x).$$

### Objective function variables:

$J(\sigma(T))$ : Objective cost functional,  $J : \Sigma \rightarrow \mathbb{R}$  and <sup>4 5</sup>

$$J(\sigma(T)) = \frac{1}{2} \sum_{k \in \frac{K}{2}} \Lambda_k Q_e \|E_k(\chi(T))\|^2 + \frac{1}{2} \|\mu(T)\|_{R_e}^2.$$

---

<sup>4</sup>Remember inner product of a complex numbers involves the conjugate transpose of one of the numbers.

<sup>5</sup>Remember the norm on a function  $y[D] \in \mathbb{L}_p^n[D]$  is  $\|y(D)\|_W^2 = \langle y(D), y(D) \rangle_W = \int_D y(t)^\top W y(t) dt$

Note:  $\frac{K}{2}$  is used because  $\Phi_k$  is discrete and  $F_k(x(T))$  is continuous, so only first half of components would line up.

$E_k(x(T))$ :  $k$ th ergodicity cost functional.  $E_k : \mathcal{X} \rightarrow \mathbb{C}$  and  $E_k(x(T)) = (F_k(x(T)) - \Phi_k)$ .

$\Lambda_k$ : Scaling factor in the Sobolev space norm of negative indices,  $\Lambda_k = \frac{1}{(1+k^\top k)^{\frac{(n+1)}{2}}}$ .

$Q_e$ : Weighting on ergodicity cost,  $Q_e \in \mathbb{R}_+$ .

$R_e$ : Weighting matrix on input effort,  $R_e \in \mathbb{R}^{m \times m}$  and  $R_e = R_e^\top \succ 0$ .

### Optimization variables:

$P(\sigma(T))$ : Projection operator,  $P : \Sigma \rightarrow \mathcal{Z}$ .

$Q_p$ : Quadratic cost of state error in projection LQ problem,  $Q_p \in \mathbb{R}^{n \times n}$  and  $Q_p = Q_p^\top \succeq 0$ .

$R_p$ : Quadratic cost of input error in projection LQ problem,  $R_p \in \mathbb{R}^{m \times m}$  and  $R_p = R_p^\top \succ 0$ .

$P_p(t)$ : Solution to algebraic Riccati equation for the projection LQ problem,  $P_p(t) \in \mathbb{R}^{n \times n}$  and  $P_p = P_p^\top \succeq 0$ .

$K_p(t)$ : Optimal feedback gain for projection LQ problem,  $K_p(t) \in \mathbb{R}^{m \times n}$ .

$\mathbb{T}_{\sigma(T)}\Sigma$ : Tangent space of trajectory manifold at the point  $\sigma(T)$  in  $\Sigma$ ,  $\mathbb{T}_{\sigma(T)}\Sigma \subset \Sigma$ .

$\zeta(T)$ : Variation in  $\sigma(T)$  and descent direction for  $J(\sigma(T))$ .  $\zeta(T) \in \mathbb{T}_{\sigma(T)}\Sigma$  and is a tuple  $\zeta(T) = (\eta(T), \nu(T))$ , where  $\eta(T) \in \mathcal{X}$  is the variation in the state trajectory and  $\nu(T) \in \mathcal{U}$  is the variation in the input trajectory.

$Q_d$ : Quadratic cost in state descent direction,  $Q_d \in \mathbb{R}^{n \times n}$  and  $Q_d = Q_d^\top \succeq 0$ .

$R_d$ : Quadratic cost in input descent direction,  $R_d \in \mathbb{R}^{m \times m}$  and  $R_d = R_d^\top \succ 0$ .

$S_d$ : Quadratic cost in final state descent direction,  $S_d \in \mathbb{R}^{n \times n}$  and  $S_d = S_d^\top \succeq 0$ .

$q_d(x(T))$ : Linear cost in state descent direction,  $q_d(x(T)) \in \mathbb{L}_2^n[T]$  and

$$q_d(x(T)) = \frac{2\pi i}{|T|} \sum_{k \in K} \Lambda_k Q_e E_k(\chi(T)) \xi_k f_k(\chi(T)).$$

$r_d(u(T))$ : Linear cost in input descent direction,  $r_d(u(T)) \in \mathbb{L}_2^m[T]$  and  $r_d(u(T)) = R_e u(T)$ .

$P_d(t)$ : Solution to quadratic terms in algebraic Riccati equation for the generalized LQ problem,  $P_d(t) \in \mathbb{R}^{n \times n}$  and  $P_d = P_d^\top \succeq 0$ .

$W_d(t)$ : Solution to linear terms in algebraic Riccati equation for the generalized LQ problem,  $W_d(t) \in \mathbb{R}^n$ .

$\gamma$ : Decent direction step size,  $\gamma \in \mathbb{R}$ .

$\epsilon$ : Threshold on when to stop optimization algorithm,  $\epsilon \in \mathbb{R}$ .

## APPENDIX B

### DERIVATIONS

#### *B.1 Generalized Linear Quadratic Problem and Solution*

The generalized linear quadratic (LQ) problem is to find the input  $u$  that minimizes the quadratic cost

$$J = \int_{t_0}^{t_f} \frac{1}{2} x^\top Q x + \frac{1}{2} u^\top R u + q^\top x + r^\top u \, dt + \frac{1}{2} x^\top(t_f) S x(t_f) \quad (202)$$

satisfying the linear system dynamics

$$\dot{x} = Ax + Bu \quad (203)$$

where

$$Q = Q^\top \succeq 0, \quad R = R^\top \succ 0, \quad S = S^\top \succeq 0. \quad (204)$$

To solve the generalized LQ problem we start with the Hamilton-Jacobi-Bellman (HJB) equation

$$-\frac{\partial J^*}{\partial t} = \min_u \left( L(x, u) + \frac{\partial J^*}{\partial x} f(x, u) \right), \quad J^*(x, t_f) = \Psi(x) \quad (205)$$

$$\begin{aligned} &= \min_u \left( \frac{1}{2} x^\top Q x + \frac{1}{2} u^\top R u + q^\top x + r^\top u \frac{\partial J^*}{\partial x} (Ax + Bu) \right), \\ J(x, t_f) &= \frac{1}{2} x^\top(t_f) S x(t_f) \end{aligned} \quad (206)$$

Let's define the argument of the minimization as

$$M = \frac{1}{2} x^\top Q x + \frac{1}{2} u^\top R u + q^\top x + r^\top u \frac{\partial J^*}{\partial x} (Ax + Bu). \quad (207)$$

Thus, to solve we take the partial of  $M$  with respect to  $u$  and set equal to zero,

$$\begin{aligned} \frac{\partial M}{\partial u} &= u^\top R + r^\top + \frac{\partial J^*}{\partial x} B = 0 \Rightarrow \\ u^* &= -R^{-1} B^\top \frac{\partial J^*}{\partial x} - R^{-1} r \end{aligned} \quad (208)$$

Plugging  $u^*$  back in to the HJB equation (205) we get

$$\begin{aligned}
-\frac{\partial J^*}{\partial t} = & \frac{1}{2}x^\top Qx + q^\top x + \\
& \frac{1}{2}(-R^{-1}B^\top \frac{\partial J^{*\top}}{\partial x} - R^{-1}r)^\top R(-R^{-1}B^\top \frac{\partial J^{*\top}}{\partial x} - R^{-1}r) + \\
& r^\top (-R^{-1}B^\top \frac{\partial J^{*\top}}{\partial x} - R^{-1}r) + \\
& \frac{\partial J^*}{\partial x} (Ax + B(-R^{-1}B^\top \frac{\partial J^{*\top}}{\partial x} - R^{-1}r)),
\end{aligned}$$

which can be simplified to

$$\begin{aligned}
-\frac{\partial J^*}{\partial t} = & \frac{1}{2}x^\top Qx - \frac{1}{2} \frac{\partial J^*}{\partial x} B R^{-1} B^\top \frac{\partial J^{*\top}}{\partial x} + \\
& \frac{\partial J^*}{\partial x} Ax - \frac{1}{2} r^\top R^{-1} r + q^\top x - r^\top R^{-1} B^\top \frac{\partial J^{*\top}}{\partial x}.
\end{aligned} \tag{209}$$

Assuming the minimal cost  $J^*$  takes on the form

$$J^*(x, t) = \frac{1}{2}x^\top Px + W^\top x + V, \quad P = P^\top \succ 0. \tag{210}$$

Taking the partial of  $J^*$  with respect to  $t$  and  $x$  we get

$$\frac{\partial J^*}{\partial t} = \frac{1}{2}x^\top \dot{P}x + \dot{W}^\top x + \dot{V} \tag{211}$$

$$\frac{\partial J^*}{\partial x} = x^\top P + w^\top. \tag{212}$$

We can then equate (209) to (211) and plug in (212) to get

$$\begin{aligned}
-\frac{1}{2}x^\top \dot{P}x + \dot{W}^\top x + \dot{V} = & \frac{1}{2}x^\top Qx - \frac{1}{2}(x^\top P + w^\top) B R^{-1} B^\top (x^\top P + w^\top)^\top + \\
& (x^\top P + w^\top) Ax - \frac{1}{2} r^\top R^{-1} r + q^\top x - r^\top R^{-1} B^\top (x^\top P + w^\top)^\top,
\end{aligned} \tag{213}$$

and remembering the the final time condition from (206),

$$J(x, t_f) = \frac{1}{2}x^\top P(t_f)x + w^\top(t_f)x + v(t_f). \tag{214}$$

Equation (213) can be simplified and divided into three differential equations

$$\begin{aligned}
-\frac{1}{2}x^\top \dot{P}x = & \frac{1}{2}x^\top Qx - \frac{1}{2}x^\top P B R^{-1} B^\top P x + \frac{1}{2}x^\top P A x + \frac{1}{2}x^\top A^\top P x \Rightarrow \\
-\dot{W}^\top x = & -w^\top B R^{-1} B^\top P x + w^\top A x + q^\top x - r^\top R^{-1} B^\top P x, \\
-\dot{V} = & -\frac{1}{2}r^\top R^{-1} r - \frac{1}{2}w^\top B R^{-1} B^\top w - r^\top R^{-1} B^\top w.
\end{aligned}$$

Factoring out the state  $x$  we a system of differential equations with the final conditions

$$\dot{P} = -PA - A^\top P - Q + PBR^{-1}B^\top P, \quad P(t_f) = S \quad (215)$$

$$\dot{W} = PBR^{-1}B^\top W - A^\top w - q + PBR^{-1}r, \quad W(t_f) = 0 \quad (216)$$

$$\dot{V} = \frac{1}{2}r^\top R^{-1}r + \frac{1}{2}w^\top BR^{-1}B^\top w + r^\top R^{-1}B^\top w, \quad V(t_f) = 0. \quad (217)$$

Plugging (212) into the optimal input,  $u^*$ , from equation (208), we get

$$u^* = -R^{-1}B^\top Px - R^{-1}(B^\top w + r). \quad (218)$$

Finally, the optimal input  $u^*$  and state  $x^*$  can be solved for by integrating the (215), (216), and (217) backwards in time from their final conditions and plugging the values into the optimal input from (218) and into the state dynamics from (203).

## ***B.2 Spatial Fourier Transform of a Trajectory***

Often the Fourier transform on a signal, or function that depends on time, is applied over the time domain. But the Fourier transform can also be applied to a trajectory, a vectored signal that depends on time, and lives in the spatial domain. To take the Fourier transform of a trajectory, the trajectory must be written as the integral of Dirac delta functions for trajectories that live in continuous domains or the sum of Kronecker delta functions for trajectories that live in discrete domains. In either case, the scaling of the delta functions should be set so norm of the trajectory equals unity.

### **B.2.1 Fourier Transform of a Continuous Trajectory Over Continuous Domain**

A continuous trajectory,  $x(T) \in \mathbb{L}_2^n[T]$ , can be written as a function of the spatial variable,  $\chi \in \mathbb{R}^n$ , as follows,

$$C(\chi; x(T)) = \frac{1}{|T|} \int_T \delta(\chi - x(t)) dt, \quad (219)$$

which has the correct scaling,

$$\begin{aligned}
\int_X C(\chi; x(T)) d\chi &= \int_X \frac{1}{|T|} \int_T \delta(\chi - x(t)) dt d\chi \\
&= \frac{1}{|T|} \int_T \int_X \delta(\chi - x(t)) d\chi dt \\
&= \frac{1}{|T|} \int_T 1 dt \\
&= \frac{1}{|T|} |T| \\
&= 1.
\end{aligned}$$

The continuous time Fourier transform is calculated by projecting the trajectory onto the Fourier basis function,  $f_k(x) = e^{2\pi i \xi_k^\top x}$ ,

$$\begin{aligned}
F_k(x(T)) &= \langle C(\chi; x(T)), f_k(\chi) \rangle \\
&= \int_X C(\chi; x(T)) \overline{f_k(\chi)} d\chi \\
&= \int_X \frac{1}{|T|} \int_T \delta(\chi - x(t)) dt \overline{f_k(\chi)} d\chi \\
&= \frac{1}{|T|} \int_T \int_X \delta(\chi - x(t)) \overline{f_k(\chi)} d\chi dt \\
&= \frac{1}{|T|} \int_T \overline{f_k(x(t))} dt.
\end{aligned}$$

### B.2.2 Fourier Transform of a Discrete Trajectory Over Continuous Domain

The discrete Fourier transform of a discrete trajectory over a continuous domain is calculated in a similar manner as the continuous Fourier transform used above,

$$D(\chi; x[K_t]) = \frac{1}{N_t} \sum_{k_t \in K_t} \delta(\chi - x[k_t])$$

with the correct scaling,

$$\begin{aligned}
\int_X D(\chi; x[K_t]) d\chi &= \int_X \frac{1}{N_t} \sum_{k_t \in K_t} \delta(\chi - x[k_t]) d\chi \\
&= \frac{1}{N_t} \sum_{k_t \in K_t} \int_X \delta(\chi - x[k_t]) d\chi \\
&= \frac{1}{N_t} \sum_{k_t \in K_t} 1 \\
&= \frac{1}{N_t} N_t \\
&= 1.
\end{aligned}$$

Finally, projecting onto the Fourier basis functions gives the answer,

$$\begin{aligned}
F_k(x[K_t]) &= \langle D(\chi; x[K_t]), f_k(\chi) \rangle \\
&= \int_X D(\chi; x[K_t]) \overline{f_k(\chi)} d\chi \\
&= \int_X \frac{1}{N_t} \sum_{k_t \in K_t} \delta(\chi - x[k_t]) \overline{f_k(\chi)} d\chi \\
&= \frac{1}{N_t} \sum_{k_t \in K_t} \int_X \delta(\chi - x[k_t]) \overline{f_k(\chi)} d\chi \\
&= \frac{1}{N_t} \sum_{k_t \in K_t} \overline{f_k(x[k_t])}.
\end{aligned}$$

### ***B.3 Decent Direction Derivation***

The decent direction used in the minimization of the cost involving ergodicity and effort can be calculated in the following manner. First, the Fourier basis functions need to be rewritten in the Taylor expansion form,

$$\begin{aligned}
e^{-2\pi i \xi_k^\top \eta(t) \varepsilon} &= \sum_{j=0}^{\infty} \frac{(-2\pi i \xi_k^\top \eta(t) \varepsilon)^j}{j!} \\
&= 1 - 2\pi i \xi_k^\top \eta(t) \varepsilon + o(\varepsilon).
\end{aligned} \tag{220}$$

Next, the Taylor expanded form the basis functions are used in the directional derivative, or Gateaux derivative, of the Fourier coefficients of the state trajectory with respect to the



variation in the state trajectory,

$$D_{\chi(T)}F_k(\chi(T)) \circ \eta(T) = \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} (F_k(\chi(T) + \varepsilon\eta(T)) - F_k(\chi)) \quad (221)$$

$$\begin{aligned} &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} \left( \frac{1}{|T|} \int_T \overline{f_k(\chi(t) + \varepsilon\eta(t))} dt - \frac{1}{|T|} \int_T \overline{f_k(\chi(t))} dt \right) \\ &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon |T|} \int_T \overline{f_k(\chi(t) + \varepsilon\eta(t))} - \overline{f_k(\chi(t))} dt \\ &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon |T|} \int_T e^{-2\pi i \xi_k^T (\chi(t) + \varepsilon\eta(t))} - e^{-2\pi i \xi_k^T \chi(t)} dt \\ &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon |T|} \int_T e^{-2\pi i \xi_k^T \chi(t)} \left( e^{-2\pi i \xi_k^T \eta(t) \varepsilon} - 1 \right) dt \\ &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon |T|} \int_T e^{-2\pi i \xi_k^T \chi(t)} (1 - 2\pi i \xi_k^T \eta(t) \varepsilon + o(\varepsilon) - 1) dt \\ &= \frac{-2\pi i}{|T|} \int_T e^{-2\pi i \xi_k^T \chi(t)} \xi_k^T \eta(t) dt \\ &= \frac{-2\pi i}{|T|} \int_T \overline{f_k(\chi(t))} \xi_k^T \eta(t) dt \\ &= \frac{-2\pi i}{|T|} (\xi_k f_k(\chi(T)) \circ \eta(T)) \Rightarrow \\ D_{\chi(T)}F_k(\chi(T)) &= \frac{-2\pi i}{|T|} \xi_k f_k(\chi(T)). \end{aligned} \quad (222)$$

Using the above directional derivative formulation the derivative of the ergodic cost with respect to the state trajectory can be calculated,

$$\begin{aligned} D_{\chi(T)} \left( \frac{1}{2} \sum_{k \in K} \Lambda_k \|E_k(\chi(T))\|^2 \right) &= \sum_{k \in K} \Lambda_k E_k(\chi(T)) D_{\chi(T)} E_k(\chi(T)) \\ &= \sum_{k \in K} \Lambda_k E_k(\chi(T)) D_{\chi(T)} (F_k(\chi(T)) - \Phi_k) \\ &= \sum_{k \in K} \Lambda_k E_k(\chi(T)) D_{\chi(T)} F_k(\chi(T)) \\ &= \sum_{k \in K} \Lambda_k E_k(\chi(T)) \frac{-2\pi i}{|T|} \xi_k f_k(\chi(T)) \\ &= \frac{-2\pi i}{|T|} \sum_{k \in K} \Lambda_k E_k(\chi(T)) \xi_k f_k(\chi(T)). \end{aligned}$$

Next, the directional derivative, or Gateaux derivative, of the effort cost with respect to the variation in the input trajectory is needed,

$$D_{\mu(T)} \left( \frac{1}{2} \|\mu(T)\|_{R_e}^2 \right) \circ \nu(T) = \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} \frac{1}{2} (\|\mu(T) + \varepsilon \nu(T)\|_{R_e}^2 - \|\mu(T)\|_{R_e}^2) \quad (223)$$

$$\begin{aligned} &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} \frac{1}{2} \left( \int_T (\mu(t) + \varepsilon \nu(t))^\top R_e (\mu(t) + \varepsilon \nu(t)) dt - \int_T \mu(t)^\top R_e \mu(t) dt \right) \\ &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} \frac{1}{2} \left( \int_T \mu(t)^\top R_e \mu(t) + \varepsilon \nu(t)^\top R_e \mu(t) + \varepsilon \mu(t)^\top R_e \nu(t) + \varepsilon^2 \nu(t)^\top R_e \nu(t) - \mu(t)^\top R_e \mu(t) dt \right) \\ &= \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} \frac{1}{2} \left( \int_T 2\varepsilon \mu(t)^\top R_e \nu(t) + \varepsilon^2 \nu(t)^\top R_e \nu(t) dt \right) \\ &= \int_T \mu(t)^\top R_e \nu(t) dt \\ &= R_e \mu(T) \circ \nu(T) \Rightarrow \\ D_{\mu(T)} \left( \frac{1}{2} \|\mu(T)\|_{R_e}^2 \right) &= R_e \mu(T). \end{aligned} \quad (224)$$

Finally, using the derivative of the ergodic cost and the the effort cost with respect to the variations of the state trajectory and input trajectory, respectively, one can calculate the derivative of the state and input trajectory tuple with respect to the variation to get the decent direction,

$$\begin{aligned} D_{\sigma(T)} J(\sigma(T)) \circ \zeta(T) &= D_{\chi(T)} \left( \frac{1}{2} \sum_{k \in K} \Lambda_k \|E_k(\chi(T))\|^2 \right) \circ \eta(T) + D_{\mu(T)} \left( \frac{1}{2} \|\mu(T)\|_{R_e}^2 \right) \circ \nu(T) \\ &= \left( \frac{-2\pi i}{|T|} \sum_{k \in K} \Lambda_k E_k(\chi(T)) \xi_k f_k(\chi(T)) \right) \circ \eta(T) + (R_e \mu(T)) \circ \nu(T). \end{aligned} \quad (225)$$

## REFERENCES

- [1] ALLAMARAJU, R., KINGRAVI, H., AXELROD, A., CHOWDHARY, G., GRANDE, R., HOW, J. P., CRICK, C., and SHENG, W., “Human aware UAS path planning in urban environments using nonstationary MDPs,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 1161–1167, IEEE, 2014.
- [2] ARMIJO, L., “Minimization of functions having Lipschitz continuous first partial derivatives,” *Pacific Journal of mathematics*, 1966.
- [3] BAGNELL, J. A. and SCHNEIDER, J., “Autonomous helicopter control using reinforcement learning policy search methods,” in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1615–1620, 2001.
- [4] BEER, R., QUINN, R., CHIEL, H., and RITZMANN, R., “Biologically inspired approaches to robotics: What can we learn from insects?,” *Communications of the ACM*, vol. 40, no. 3, pp. 30–38, 1997.
- [5] BROCKETT, R. W., “Asymptotic Stability and Feedback Stabilization,” *Differential Geometric Control Theory*, pp. 181–191, 1983.
- [6] BROOKS, A., MAKARENKO, A., WILLIAMS, S., and DURRANT-WHYTE, H., “Planning in continuous state spaces with parametric POMDPs,” in *IJCAI Workshop Reasoning with Uncertainty in Robotics*, 2005.
- [7] BROOKS, R., “I, Rodney Brooks, Am a Robot,” June 2008.
- [8] BRYSON, J. A. E. and HO, Y.-C., *Applied Optimal Control: Optimization, Estimation and Control*. Taylor & Francis, revised ed., Jan. 1975.
- [9] CAMPI, M. C. and VIDYASAGAR, M., “Learning with prior information,” *Automatic Control, IEEE Transactions on*, vol. 46, pp. 1682–1695, Sept. 2001.
- [10] CHOSET, H., LYNCH, K. M., HUTCHINSON, S., KANTOR, G. A., BURGARD, W., KAVRAKI, L. E., and THRUN, S., *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, the mit press ed., May 2005.
- [11] CRAWFORD, L. and SASTRY, S., “Learning Controllers for Complex Behavioral Systems,” Tech. Rep. UCB/ERL M96/73, University of California at Berkeley, Dec. 1996.
- [12] DROGE, G. and EGERSTEDT, M., “Multi-Modal Parametrized MPC for Mobile Robot Navigation,” *IEEE Transactions on Control Systems Technology*.

- [13] EGERSTEDT, M. and MARTIN, C., *Control Theoretic Splines: Optimal Control, Statistics, and Path Planning (Princeton Series in Applied Mathematics)*. Princeton University Press, Dec. 2009.
- [14] ELFES, A., “Using Occupancy Grids for Mobile Robot Perception and Navigation,” *Computer*, vol. 22, pp. 46–57, June 1989.
- [15] FREW, E., “Receding horizon control using random search for UAV navigation with passive, non-cooperative sensing,” in *AIAA Guidance, Navigation, and Control Conference*, (San Francisco, CA), 2005.
- [16] GOEBEL, R., SANFELICE, R., and TEEL, A., “Hybrid dynamical systems,” *IEEE Control Systems Magazine*, vol. 29, pp. 28–93, Apr. 2009.
- [17] HAUSER, J., “A projection operator approach to the optimization of trajectory functionals,” *IFAC world congress*, 2002.
- [18] HESTER, T., QUINLAN, M., and STONE, P., “Generalized model learning for reinforcement learning on a humanoid robot,” in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2369–2374, IEEE, 2010.
- [19] HINNEBURG, A. and KEIM, D., “Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering,” in *Proceedings of 25th International Conference on Very Large Data Bases, VLDB*, pp. 506–517, 1999.
- [20] HUSSEIN, I. I. and STIPANOVIC, D. M., “Effective Coverage Control for Mobile Sensor Networks With Guaranteed Collision Avoidance,” *Control Systems Technology, IEEE Transactions on*, vol. 15, no. 4, pp. 642–657, 2007.
- [21] ISLAM, S. and LIU, P. X., “Adaptive iterative learning control for robot manipulators without using velocity signals,” in *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*, pp. 1293–1298, IEEE, July 2010.
- [22] ISO/IEC, *Part 9126-1:2001: Software engineering – Product quality – Part 1: Quality model*. International Organization for Standardization, Geneva, Switzerland., 18. June 2011.
- [23] KAEHLING, L., LITTMAN, M., and MOORE, A., “Reinforcement learning: A survey,” *CoRR*, vol. cs.AI/9605103, 1996.
- [24] KHALIL, H., “Nonlinear systems.” Prentice Hall, 2002.
- [25] KIM, Y., KIM, S. H., and KWAK, Y. K., “Dynamic Analysis of a Nonholonomic Two-Wheeled Inverted Pendulum Robot,” *Journal of Intelligent and Robotic Systems*, vol. 44, pp. 25–46, Jan. 2006.
- [26] KOBER, J. and PETERS, J., “Imitation and Reinforcement Learning,” *Robotics & Automation Magazine, IEEE*, vol. 17, no. 2, pp. 55–62, 2010.

- [27] KOENIG, S. and SIMMONS, R. G., “Complexity Analysis of Real-Time Reinforcement Learning,” in *Conference on Artificial Intelligence*, pp. 99–105, 1993.
- [28] KOHL, N. and STONE, P., “Policy gradient reinforcement learning for fast quadrupedal locomotion,” in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2619–2624, 2004.
- [29] KOLLAR, T. and ROY, N., “Trajectory optimization using reinforcement learning for map exploration,” *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 175–196, 2008.
- [30] KOLTER, J. Z., JACKOWSKI, Z., and TEDRAKE, R., “Design, analysis and learning control of a fully actuated micro wind turbine,” in *Proceedings of the 2012 American Control Conference (ACC)*, pp. 2256–2263, June 2012.
- [31] KUO, F. and SLOAN, I., “Lifting the curse of dimensionality,” *Notices of the AMS*, vol. 52, no. 11, pp. 1320–1328, 2005.
- [32] LEUNG, C., HUANG, S., KWOK, N., and DISSANAYAKE, G., “Planning under uncertainty using model predictive control for information gathering,” *Robotics and Autonomous Systems*, vol. 54, pp. 898–910, Nov. 2006.
- [33] LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W., and KAREL, C., “An Algorithm for the Traveling Salesman Problem,” *Operations Research*, vol. 11, no. 6, pp. 972–989, 1963.
- [34] MATHEW, G. and MEZIC, I., “Metrics for ergodicity and design of ergodic dynamics for multi-agent systems,” *Physica D*, vol. 240, pp. 432–442, Feb. 2011.
- [35] MCENEANEY, W., “Curse-of-dimensionality free method for Bellman PDEs with Hamiltonian written as maximum of quadratic forms,” in *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference (CDC-ECC)*, pp. 42–47, IEEE, 2005.
- [36] MERRIAMWEBSTER.COM, “learn.” <http://www.merriam-webster.com/dictionary/learn>, June 2012. Web. 17 September 2012.
- [37] MILLER, L. M. and MURPHEY, T. D., “Trajectory optimization for continuous ergodic exploration,” in *American Control Conference (ACC), 2013*, pp. 4196–4201, IEEE, 2013.
- [38] MILLER, L. M., SILVERMAN, Y., MACIVER, M. A., and MURPHEY, T. D., “Ergodic Exploration of Distributed Information,” *IEEE Transactions on Robotics*, pp. 1–12.
- [39] MILLER, L. M. and MURPHEY, T. D., “Optimal Planning for Target Localization and Coverage Using Range Sensing,” in *IEEE International Conference on Robotics and Automation ICRA*, pp. 1–8, 2015.

- [40] MITCHELL, T. M., *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1 ed., Mar. 1997.
- [41] MOORE, K. L., CHEN, Y. Q., and AHN, H.-S., “Iterative Learning Control: A Tutorial and Big Picture View,” in *Decision and Control, 2006 45th IEEE Conference on*, pp. 2352–2357, 2006.
- [42] MOZER, M. C. and BACHRACH, J., “Discovering the Structure of a Reactive Environment by Exploration,” *dx.doi.org*, Mar. 1990.
- [43] MUSSA-IVALDI, F., GISZTER, S., and BIZZI, E., “Linear combinations of primitives in vertebrate motor control,” *Proceedings of the National Academy of Sciences*, vol. 91, no. 16, p. 7534, 1994.
- [44] OUYANG, P. R. and PIPATPAIBUL, P., “Iterative Learning Control: A Comparison Study,” *ASME 2010 International Mechanical Engineering Congress and Exposition*, pp. 939–945, Nov. 2010.
- [45] OUYANG, P. R., ZHANG, W. J., and GUPTA, M. M., “An adaptive switching learning control method for trajectory tracking of robot manipulators,” *Mechatronics*, vol. 16, no. 1, pp. 51–61, 2006.
- [46] PAPADIMITRIOU, C. H., “Computational Complexity,” in *Encyclopedia of Computer Science*, pp. 260–265, Chichester, UK: John Wiley and Sons Ltd., 2003.
- [47] SASTRY, S., *Nonlinear systems. analysis, stability, and control*, Springer Verlag, June 1999.
- [48] SATO, M., ABE, K., and TAKEDA, H., “Learning control of finite Markov chains with an explicit trade-off between estimation and control,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 18, no. 5, pp. 677–684, 1988.
- [49] SETHIAN, J. A., “Fast marching methods,” *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.
- [50] SILVERMAN, Y., MILLER, L. M., MACIVER, M. A., and MURPHEY, T. D., “Optimal planning for information acquisition,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 5974–5980, IEEE, 2013.
- [51] SPONG, M. W., HUTCHINSON, S., and VIDYASAGAR, M., *Robot modeling and control*. John Wiley & Sons, Inc., 2006.
- [52] STULP, F., THEODOROU, E., KALAKRISHNAN, M., PASTOR, P., RIGHETTI, L., and SCHAAL, S., “Learning motion primitive goals for robust manipulation,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 325–331, IEEE, 2011.
- [53] SUN, H., HOU, Z., and LI, D., “Coordinated Iterative Learning Control Schemes for Train Trajectory Tracking With Overspeed Protection,” *Automation Science and Engineering, IEEE Transactions on*, vol. 10, pp. 323–333, Apr. 2013.

- [54] SUTTON, R. and BARTO, A., *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, The MIT Press, the mit press ed., Mar. 1998.
- [55] THEODOROU, E., BUCHLI, J., and SCHAAAL, S., “A Generalized Path Integral Control Approach to Reinforcement Learning,” *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, Mar. 2010.
- [56] THOROUGHMAN, K. and SHADMEHR, R., “Learning of action through adaptive combination of motor primitives,” *Nature*, vol. 407, no. 6805, p. 742, 2000.
- [57] THRUN, S., LIU, Y. F., KOLLER, D., NG, A. Y., GHAHRAMANI, Z., and DURRANT-WHYTE, H., “Simultaneous localization and mapping with sparse extended information filters,” *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.
- [58] THRUN, S., “Efficient Exploration in Reinforcement Learning,” Tech. Rep. CMU-CS-92-102, School of Computer Science Carnegie Mellon University, Pittsburgh, Pennsylvania, Jan. 1992.
- [59] THRUN, S. B. and MÖLLER, K., “Active Exploration in Dynamic Environments,” pp. 531–538, 1992.
- [60] VALIANT, L. G., “A theory of the learnable,” in *STOC ’84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, ACM Request Permissions, Dec. 1984.
- [61] VAPNIK, V. N. and CHERVONENKIS, A. Y., “On the uniform convergence of relative frequencies of events to their probabilities,” *Theory of Probability and its Applications*, vol. 16, no. 2, pp. 264–280, 1971.
- [62] VIDYASAGAR, M. and KARANDIKAR, R. L., “A learning theory approach to system identification and stochastic adaptive control,” *Journal of Process Control*, vol. 18, pp. 421–430, Mar. 2008.
- [63] VIDYASAGAR, M., *Learning and Generalization: With Applications to Neural Networks (Communications and Control Engineering)*. Springer, softcover reprint of hardcover 2nd ed. 2002 ed., Dec. 2010.
- [64] VINCENT, L. and SOILLE, P., “Watersheds in digital spaces: an efficient algorithm based on immersion simulations,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, pp. 583–598, June 1991.
- [65] WEI, D. and PANAITESCU, R., “An implementation of iterative learning control in industrial production machines,” in *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, pp. 472–477, Aug. 2008.
- [66] WEI, Q. and LIU, D., “A Novel Iterative  $\theta$ -Adaptive Dynamic Programming for Discrete-Time Nonlinear Systems,” *Automation Science and Engineering, IEEE Transactions on*, no. 99, pp. 1–15, 2013.

- [67] WIEWIORA, E., “Efficient Exploration for Reinforcement Learning,” 2004.
- [68] WIKIPEDIA.COM, “Machine learning.” [http://en.wikipedia.org/wiki/Machine\\_learning](http://en.wikipedia.org/wiki/Machine_learning), June 2012. Web. 17 September 2012.
- [69] ZHANG, W. J., OUYANG, P. R., and SUN, Z. H., “A novel hybridization design principle for intelligent mechatronics systems,” in *Proceedings of International Conference on Advanced Mechatronics*, pp. 4–6, 2010.
- [70] ZOPPOLI, R., SANGUINETI, M., and PARISINI, T., “Can we cope with the curse of dimensionality in optimal control by using neural approximators?,” in *Proceedings of the 40th IEEE Conference on Decision and Control*, pp. 3540–3545, IEEE, 2001.